HEWLETT PACKARD

HP 37900D
Signaling Test Set
Programming Manual

HP 37900D Signaling Test Set

Programming Manual

HP Part No. 37900-90090
Printed in U.K. October 1994

Edition 1 - Software Revision A06.50

Notice

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

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Printing History

First Edition October 1994

WARRANTY

This Hewlett-Packard product is warranted against defects in materials and workmanship for a period of one year from date of shipment. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products which prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by HP. Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country.

HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

LIMITATION OF WARRANTY

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the products, or improper site preparation or maintenance.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

EXCLUSIVE REMEDIES

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HP SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

CERTIFICATION

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility and to the calibration facilities or other International Standards Organization members.

ASSISTANCE

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your Hewlett-Packard Sales and Service Office. Addresses are provided at the back of this manual.

WARNING

READ THE FOLLOWING NOTES BEFORE INSTALLING OR SERVICING ANY INSTRUMENT.

1. IF THIS INSTRUMENT IS TO BE ENERGISED VIA AN AUTO-TRANSFORMER MAKE SURE THAT THE COMMON TERMINAL OF THE AUTO-TRANSFORMER IS CONNECTED TO THE NEUTRAL POLE OF THE POWER SOURCE.
2. THE INSTRUMENT MUST ONLY BE USED WITH THE MAINS CABLE PROVIDED. IF THIS IS NOT SUITABLE, CONTACT YOUR NEAREST HP SERVICE OFFICE. THE MAINS PLUG SHALL ONLY BE INSERTED IN A SOCKET OUTLET PROVIDED WITH A PROTECTIVE EARTH CONTACT. THE PROTECTIVE ACTION MUST NOT BE NEGATED BY THE USE OF AN EXTENSION CORD (POWER CABLE) WITHOUT A PROTECTIVE CONDUCTOR (GROUNDING).
3. BEFORE SWITCHING ON THIS INSTRUMENT:
   a. Make sure the instrument input voltage selector is set to the voltage of the power source.
   b. Ensure that all devices connected to this instrument are connected to the protective (earth) ground.
   c. Ensure that the line power (mains) plug is connected to a three-conductor line power outlet that has a protective (earth) ground. (Grounding one conductor of a two-conductor outlet is not sufficient).
   d. Check correct type and rating of the instrument fuse(s).

How to Use This Manual

The Programming Manual provides guidance on developing tests to check network performance. It is organized in seven main chapters:

Chapter 1 summarizes the files which need to be considered in developing a test.

Chapter 2 describes how to customize the Autostart File.

Chapter 3 describes Personality Files and how to customize them.

Chapter 4 describes the files involved in creating or modifying an Application.

Chapter 5 describes the operation of Test Sequences and provides the information needed to create or modify a Test Sequence.

Chapter 6 describes how to create a signaling message using a Message Input File.

Chapter 7 provides information on decodes provided by Hewlett-Packard, and describes how to develop decodes.

Contents

Figures

Tables

1

Overview

The following table summarizes the files used to govern the test capabilities of the HP 37900.

User File Summary

Autostart file	Ensures that the necessary files are loaded when the HP 37900 boots up.	Chapter 2
Personality File	Customizes the HP 37900 to the signaling system being tested. Enables/disables some optional facilities.	Chapter 3
Applications File	Shows the test applications available with its associated personality file. Contains the test procedures which run the application tests.	Chapter 4
Application CALL Procedures	Called from the test procedures in the Application Control file.	Chapter 4
Test Sequence	Controls all activities in an Emulation Test.	Chapter 5
Emulation CALL Procedures	Called from within a test sequence during its execution. Enables activities not otherwise available in a test sequence.	Chapter 5

2

Autostart File

When the HP 37900 is switched on (or rebooted), the AUTOSTART file is used to configure the software.

This includes permanently loading user files such as point code files, real-time decodes and CALL procedures.

Use the File Manager editor to edit this file as required. In the following example, a line has been inserted to permanently load the new real-time decode RTDNEW. If there are real-time decodes you don't need, "comment them out" by placing a * as the first character in the line.

```
*
* Autostart File Rev a.03.00
*

Define SYS: as system volume (*) and STSS: as default (:)
*
W
SSYS:
DSTSS:
Q
*
*Execute COPY_OFF to ensure I-Cache performs a copy-back
X*COPY_OFF
*
* Install printer dump utility and enable
*   printer type [THINKJET|QUIETJET]
*
P*:DUMPG
X*:DUMPG
THINKJET
```

```
*Permanently load required modules of test set software
* |
* +-- Signaling Test Set Library
* |
P*:STSLIB
*
* |
* +-- load filer
* |
P*FUTIL
* |
* +-- load editor
* |
P*STSED
* |
* +-- initialise filer and editor
* |
P*STSISRS
X*INSTISRS
X*EDINIT
* |
* +-- User-defined point code methods
* |
P*:PCCITT
P*:PCANSI
P*:PCG7
P*:PCCINT
P*:PCCNAM
P*:PCANAM
P*:PCGNAM
P*:PCJPN
* |
* +-- Real Time Decode modules
* |
P*RTDSCROLL
* Real Time Decode scroll library
*
```

```
P*RTDSPLIT
P*RTDSTEL
P*RTDSTEL2
P*RTDCOL
P*RTD7STAT
P*RTDISTAT
P*RTDBSTAT

P*RTDNEW
```
Insert this line to load RTDNEW at bootup

```
     |
     |
     |
* CALL procedures for user-defined Swiss and TRAU call trace
P*:SWISSCT
P*:TRAUCALL
*
* Execute test set main menu
*
X*HINIT
X*STS
```

3

Personality and Personality Files

Introduction

The decoding rules adopted by the HP 37900 when defining logging conditions, searching and decoding logged data or creating messages are governed by the assigned personality.

This chapter describes how to ensure that the correct personality file is active and that the correct personality is assigned to the channels you are going to use. It then describes what personalities and personality files consist of, and how to create your own versions.

Personality

A personality defines the default decodes and alternative decodes that are to be applied to the signaling data in either Monitor Mode or Emulation Mode. In Monitor Mode, the personality also defines the default call trace files, protocols and mnemonics used, and which real-time decodes are available.

3

A personality is assigned to each channel (Monitor Mode) and each link (Emulation Mode) by the active personality file (see below). You can change the personality assignment from the MONITOR MODE menu (`A - change channel Attributes`) or the EMULATION MODE menu (`A - Assign personality`).

The Active Personality File

The active personality file completely defines the two personalities currently being used by the HP 37900, and specifies the channels and links to which each is assigned. You can replace the default personality file, as described below.

The entire operation of the HP 37900 is governed by the active personality file, so when you select a personality file, ensure that its definitions match your requirements.

A number of personality files have been provided with the HP 37900. User-defined personality files can be added, as described later in this chapter. Use the File manager facility to list the personality files included in your HP 37900.

Although any number of personality files can be stored on disc, only one can be the currently active personality file. When the HP 37900 boots up, it loads PERFILE.P as the default personality file. During software installation, a user-response decides whether this is a copy of the CCITT personality file CCITTRB.P or for a North American system ANSI8588.P, or a user-modified version of one of these.

You can specify a different default personality file. Use the File Manager and editor to edit PERFILE.P or overwrite it with another personality file.

You can also temporarily load a different personality file (option `P` from the HP 37900 Main Menu). This will remain active until either the HP 37900 is rebooted or you select a different file.

Personality Assignment

You can see quickly which of the two personalities in the active personality file has been assigned to each link or channel, and if necessary change the assignment.

In Monitor Mode;
Select A from the Monitor Mode menu.

In Emulation Mode;
Select A from the Emulation Mode menu.

HP-Supplied Personality Files

The personality files listed below are provided by Hewlett-Packard. The list shows the name of each personality file and the standards (and names) of the personalities they contain.

The first in the list, CCITTRB.P is referred to as the "standard" European personality file. ANSI8588.P is the "standard" North American personality file.

CCITTRB.P	Personality 1 = CCITT Red Book (REDBOOK) Personality 2 = CCITT Blue Book (BLUEBOOK)
CCITTBW.P	Personality 1 = CCITT Blue Book (BLUEBOOK) Personality 2 = CCITT White Book (WHITEBOOK)
BELLAN88.P	Personality 1 = Bellcore (BELL87) Personality 2 = ANSI 1988 (ANSI88)
BELLCCR.P	Personality 1 = Bellcore 1987 (BELL87) Personality 2 = CCITT Red Book (REDBOOK)
ANS88CCR.P	Personality 1 = ANSI 1988 (ANSI88) Personality 2 = CCITT Red Book (REDBOOK)
ANS85CCR.P	Personality 1 = ANSI 1985 (ANSI85) Personality 2 = CCITT Red Book (REDBOOK)
ANSI8588.P	Personality 1 = ANSI 1985 (ANSI85) Personality 2 = ANSI 1988 (ANSI88)
CCRBF8.P	Personality 1 = CCITT Red Book + Finnish TUP (REDBOOK) Personality 2 = CCITT Blue Book + Finnish TUP (BLUEBOOK)

G7CCR.P	Personality 1 = GERMAN 1TR7 1987 spec.GER1TR7) Personality 2 = CCITT Red Book (REDBOOK)
CCRBNM89.P	Personality 1 = CCITT Red Book + Nordic Mobile Telephone (HUP) + Finnish TUP (RED+NMT) Personality 2 = CCITT Blue Book + Nordic Mobile Telephone (HUP) + Finnish TUP (BLUE+NMT)
ABISTRAU.P	Personality 1 = ISDN ABIS27 Q.931 (BLUEISDN) Personality 2 = ISDN G0860 (TRAU)
ABISGSM.P	Personality 1 = ISDN (BLUEISDN) Personality 2 = CCITT Blue Book (BLUEBOOK)
CCRBBT.P	Personality 1 = CCITT Red Book + BT National User Part (REDBOOK) Personality 2 = CCITT Blue Book + BT National User Part (BLUEBOOK)
SWISS.P	Personality 1 = CCITT Blue Book ISDN Q.931 (DSS1CH) Personality 2 = CCITT Blue Book - Swiss PTT versions (SS#7CH)

Note: CCITTRB.P includes calls to GSM decodes (MAP, DTAP, BSSMAP).

Contents of a Personality File

To inspect a personality file currently stored in your HP 37900;

1. Enter File Manager.
2. List the files in the STSS volume by Type.
3. Select Personality Files.
4. Highlight the personality file you want to look at.
5. Press E.

The following list is based on the HP-supplied personality file CCITTRB. It shows that a personality file completely defines its two personalities in terms of the following types of information. See the notes following the list for further information on the items highlighted by **bold print**.

- The **Application** command **APP**.
- The personality **TITLE** command.

- The **names** of the two personalities defined in the file.
- The numbers and names identifying the available **real-time decodes**.
- Assignment of personality to signaling channels (and to signaling links for Emulation Mode).
- The Point-code style (CCITT 14-bit or ANSI 24-bit) assigned to each personality.
- **Point Code Display formatter file names.**
- Call Trace files.
- **Reset Override (RO)** flag.
- **RTD Overflow Warning message (RTO)** flag.
- **Alternative Start Decode (SD)** flag.
- The names of the **decodes** assigned for LAP, MTP, Service part and Userdata part.
- The names of the **keystroke-selected** decodes for MTP, Service Part and Userdata Part, and the keys which select them.
- The **mnemonics** defined for each personality for:
  - Service Indicator (SI)
  - Sub-service Field (SF)
  - Message Type (MT)
  - Header codes H0 / H1 (HH)

  and for ISDN personalities the mnemonics for each:
  - Call Control Message Type.
  - Management Entity Identity Message Type.
- H0/H1 field length.
- **Hex Decode (HD).**
- **Dual.**

Notes on File Contents

These notes correspond to the items, **highlighted by bold print**, in the list on the previous page.

- The **APP** command enables Applications mode of operation. When used with a personality name (for example `APP ABISGSM`) it adopts the set of applications associated with that personality. The **TITLE** command displays the name of the personality. See Chapter 1.
- The **names** should represent the personalities contained in the file. Maximum 8 alphanumeric characters.
- There can be up to 8 **real-time decodes**.

  For software earlier than Revision 05.50, this section also defines upper and lower display thresholds, which limit the queue of messages waiting to be displayed during real-time monitoring. For Revision 05.50 and above, these fields are not required and are ignored.

  This section also shows the real-time decodes which can be used to format a logged-data print.

- **Point Code Display formatter file names**. To enable you to specify OPC and DPC by user-defined mnemonics, the Point Code Display formatter file accesses its associated mnemonics file.
  PCANAM accesses **PCA.TEXT** (ANSI)
  PCCNAM accesses **PCC.TEXT** (CCITT)
  PCGNAM accesses **PCG.TEXT** (German 1TR7)
  Use the File Manager and editor to ensure the mnemonics file exists and the mnemonics you want to use are defined in it.

  To create the mnemonics file;

  Hewlett-Packard have provided three sample mnemonic files;

  PCADEMO.TEXT (ANSI)
  PCCDEMO.TEXT (CCITT)
  PCGDEMO.TEXT (German 1TR7)

  These files explain the format and syntax required, and contain a number of demonstration example mnemonics.

Copy one of these into the equivalent mnemonics file. For example, copy PCCDEMO.TEXT into PCC.TEXT.

Edit the file to add the mnemonics you want.

- **Reset Override (RO)**. With this enabled, when you return to the level 2/3 decode screen from the MONITOR MODE menu, the same information is displayed as when it was last viewed. If it is not enabled, the start of channels 1 and 2 are displayed.

- **RTD Overflow Warning message (RTO)**. Set ON or OFF (default OFF).

  If messages are being received at a high rate, it is possible that some are not displayed in real-time (but are logged). The following warning message can be displayed if this happens.
  `Warning!! Real-time decode overflow occurred`

  RTO ON allows the message to be displayed.

  RTO OFF suppresses the message.

- **Alternative Start Decode (SD)**. `SD ON` defines that the decode display will start at a specified octet within the message, instead of the first octet. The start octet is defined within the decode.

- Up to 3 available **decodes** for LAP, MTP, each defined SI, and each defined SSN.

- Up to 16 **keystroke-selected** decodes for each personality file.

- The **mnemonics** for the HH and MT entries MUST be in alphabetic order.

- **Hex Decode (HD)**. This allows you to define that the octet values in the message decode and field decode will be displayed in hexadecimal, instead of binary.

- **Dual**. If this command is present, a complete call trace can be performed, on for example Abis and "A" interfacing or ISDN and SS7 links, using two personalities simultaneously.

Limits

Maximum mnemonics per HP 37900:

SI - 16
SSF- 16
MT - 100
HH - 200

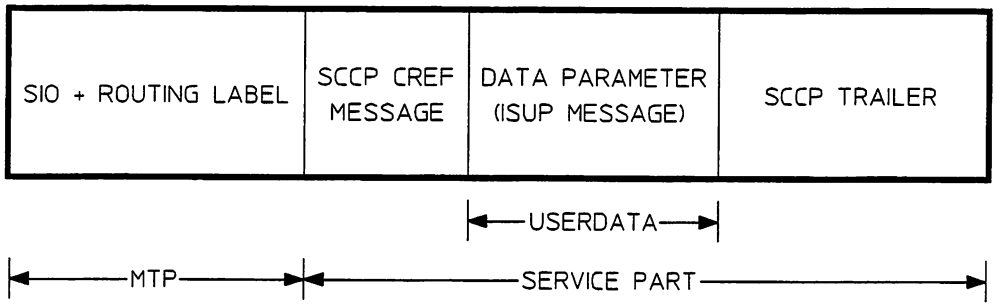
Maximum decode entries per personality (each including up to 3 decodes):

Service Decodes (SV) - 16
UserData Decodes (UD)- 16

Personality Control of a Decoding Sequence (Example)

The message decoding sequence has up to three parts; decoding the Message Transfer Part (MTP), the Service Part, and (if required) the UserData part. This example shows how these combine to produce a complete message decode.

Consider decoding an SCCP CREF message, containing an ISUP message in its Data parameter. This is one of the more complex messages. The CREF message is defined in the CCITT Red Book, Q.713 section 4.4. 3



Assume that Personality 1 is CCITT Redbook, and that the personality file includes the following lines :

MP 1 *CRMTP*

SV 1 3 XXXX *CRSCCP*

UD 1 3 *CRISUP*

1. The first line shows that Personality 1 assigns the decode *CRMTP* to decode the SIO and Routing Label (the Message Transfer Part).

   MP — shows that the decode being assigned in this step is for the Message Transfer Part (MTP).

   1 — identifies which personality (1 or 2) in the active personality file assigns the decode *CRMTP*.

   CRMTP — is the name of the assigned decode.
   C denotes CCITT.
   R denotes Red Book version.
   MTP Message Transfer Part.

2. The second line shows that Personality 1 assigns *CRSCCP* to decode the Service Part of a message whose SI value is 3.

   SV — shows that the decode being assigned in this step is for the Service Part (otherwise known as the User Part).

   1 — identifies the personality.

   3 — When assigning the Service Part decode, the number in this column denotes the SI value.

   XXXX — is the Sub-Service Field (SSF) value. Each bit can have the value 0, 1, or X (don't care). In this example, all bits are set to X, so the decode is determined by the SI value alone.

   CRSCCP — is the name of the assigned decode (see the explanation for *CRMTP* above).

   Up to two alternative decodes could be assigned.

3. The third line shows that Personality 1 assigns *CRISUP* to decode the UserData Part of a message whose SSN value is 3.

   UD — shows that the decode being assigned in this step is for the UserData Part.

   1 — identifies the personality.

   3 — When assigning the UserData Part decode, the number in this column denotes the Sub-Service Number (SSN), which identifies the user function (ISUP in this example).

CRISUP is the name of the assigned decode.

Up to two alternative decodes could be assigned.

4. When the UserData has been decoded, it is still necessary to decode the SCCP trailer. The *CRISUP* decode is recalled for this. 3

A simplified picture of the interactions is shown below.

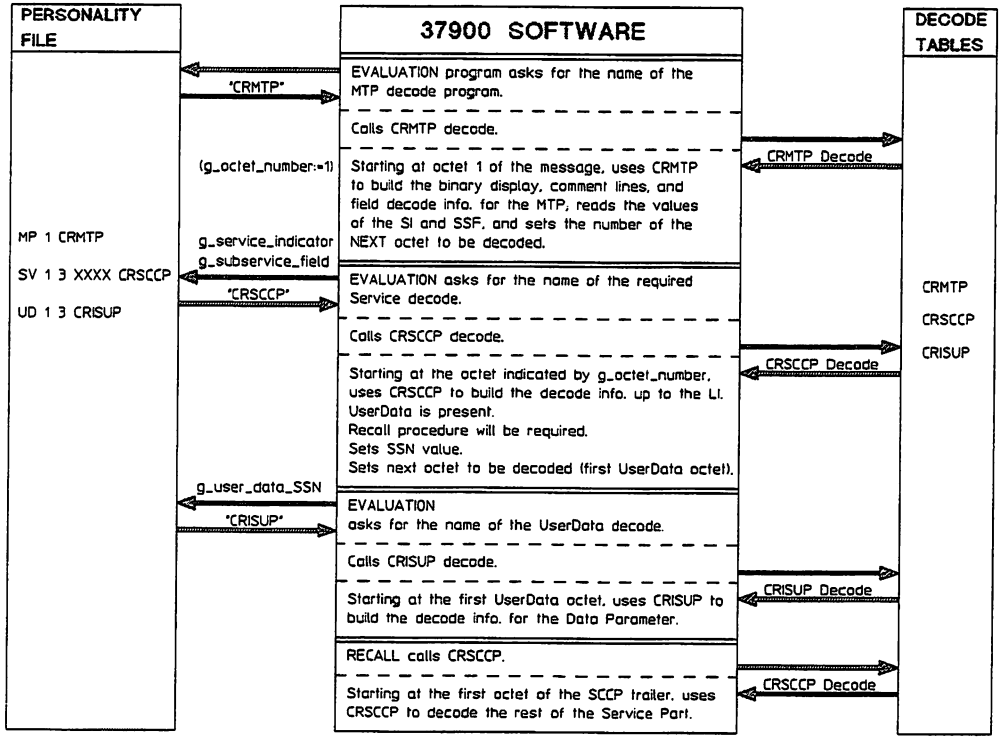


Figure 3-1. Decode Sequence

To Modify or Construct a Personality File

Whether you want to make a small change to an existing file, or create a completely new file, the method is the same. That is, edit an existing file, save it to disc under a suitable name, record its name and function.

3 The File Manager mentioned below is described in the "FILE MANAGER and Editor" chapter.

Refer to the list of personality files under "HP-Supplied Personality Files", or use the File Manager to produce a list of available personality files. Select the file closest to your needs.

Use the editing facility within File Manager to create your file and either save it to a new filename which describes its function or overwrite the existing file.

4

Applications

Overview

Operation of the HP 37900 in applications testing is governed by the active *Application File*.

The Application File is activated directly by a command in the active personality file.

Hewlett-Packard provide an Application File for every personality file supplied with the product.

Personality File APP Commands

- `APP` (with no Application File specified)
  - Enables the initial applications startup screen and personalities menu.
  - Activates the default Application File `APPFILE.K`.
- `APP <filename.K>`
  - Enables the initial applications startup screen and personalities menu.
  - Activates the Application File `<filename.K>`.

  For example, the personality file `ABISGSM.P` contains the commands
  `APP ABISGSM`
  to activate the Application file `ABISGSM.K`.

APPFILE.K

This file:

- Sets up the Manual Mode menu (`MENU3`) and contains the procedures required to start the Manual Mode operations.

- Sets up the Help menu (MENU4) and contains the procedure required to start the Help facility.

The Applications menu and procedures are omitted in the original version of APPFILE.K as provided by Hewlett-Packard. If you want to change this, see under "Changing the Default Application File".

Note

The personalities menu (MENU1) is set up automatically by the HP 37900 by scanning the discs for Personality files.

4

Named Application File (filename.K)

For each personality file provided by Hewlett-Packard there is an associated Application File.

In addition to the functions described above for APPFILE.K, these files set up the Applications menu (MENU2) appropriate for its personality, and contain the procedures required to start each application.

Changing the Default Application File

If you want the HP 37900 to adopt a particular set of applications at startup, you can either:

- Ensure the default personality file activates the Application File you want.
- Change the default Application File APPFILE.K. First save the original version of APPFILE.K to a different filename, then either;
  - Overwrite APPFILE.K with the Application File you want.
  - Edit APPFILE.K to include the applications you want.

The Application File

The Application File has two main sections. The first section sets up the menus, each menu option representing an application. The second section contains the procedures required to run the applications.

The selected menu option acts as an entry point to its associated procedure.

For most monitoring applications, the accessed procedure starts monitoring and uses an Application CALL procedure to format the monitor display.

For example, the Application File ABISGSM.K includes the following entries.

In the menu section:

```
*
*
*     Menu number.
*    |
*    |    _Application title maximum 15 chars.
*    |   |
*    |   |
*    V   V
*
APP 2   RTDGSM
"Press RETURN for real-time decode of 'A' and 'A-bis' messages."
APP 2   RTDGSM1
"Press RETURN for partial real-time decode of 'A' and 'A-bis' messages."

        |
        |
```

The line "APP 2 RTDGSM" specifies RTDGSM as the name of the first application in the Applications menu (MENU2). (The next line appears at the bottom of the display when you highlight this option.)

In the procedures section:

```
       |
       |
*
*
RTDGSM                                          Start here when RTDGSM is
                                                selected from the Applications
                                                menu.
* Monitor the signaling link using the first real-time decode
M                                               Enter Monitor Mode.
#CALL APPLIB_RTD1                               Invoke the CALL procedure
                                                which starts and controls the
                                                test.
!                                               End of the procedure.
RTDGSM1
* Monitor the signaling link using the second real-time decode
M
#CALL APPLIB_RTD1
!
       |
       |
```

The comment lines refer to the list of real-time decodes (RTDs) specified in the personality file.

Editing the Applications File

This section uses an example to show how to add an application to the HP 37900.

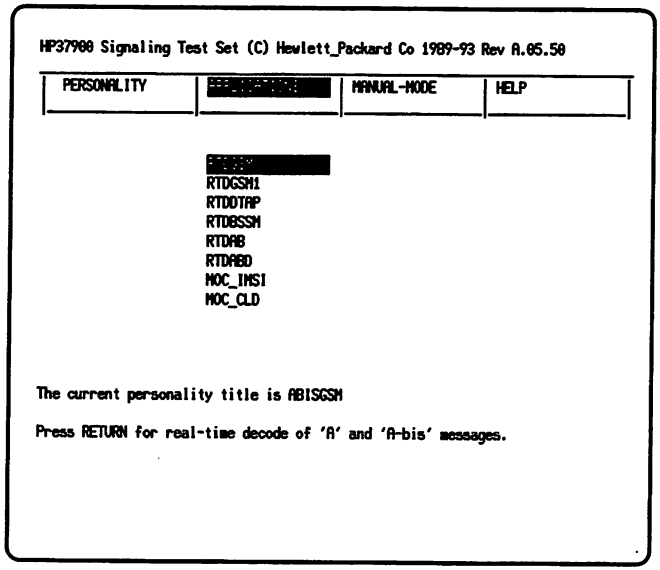
The tasks involved are:

- Insert the application name in the menu section of the Applications file.
- Insert the procedure in the Applications file procedure section.
- If necessary, create the Application CALL Procedure and add it to the Application CALL procedure library APPLIB.

4

Example

To add the new application RTDNEW to the ABISGSM personality.



Use the Pascal Editor to edit the file ABISGSM.K. (remember to add the final period).

1. Set up the menu option.

```
*************************************************************************
* Filename : ABISGSM.K
* Revision 1.0
*************************************************************************

*
*  HP 37900 APPLICATION FILE
*
*************************************************************************
*
* The first section sets up the menu titles, and adds a corresponding
* line of text for each title.
*
*
*     -------------------------------Menu number 1, 2, 3 or 4
*    |  ---------------------------- Menu TiTle Maximum 15 characters
*    | |
*    | |
*    V V
MENU 1 Personality
"Cursor keys are active ( DOWNCURSOR lists personalities)."
MENU 2 Applications
"Cursor keys are active ( DOWNCURSOR lists applications)."
MENU 3 Manual-Mode
"Cursor keys are active ( DOWNCURSOR lists manual options)."
MENU 4 Help         "Press [Select] to obtain help"
"Cursor keys are active ( DOWNCURSOR lists help options)."
*
*
*
* This section sets up the menu options and their descriptive text.
*
*
*     Menu number.
*   |
*   |   _Application title maximum 15 chars.
*   |  |
*   |  |
*   V  V
*
APP 2  RTDGSM
"Press RETURN for real-time decode of 'A' and 'A-bis' messages."
APP 2  RTDGSM1
"Press RETURN for partial real-time decode of 'A' and 'A-bis' messages."
APP 2  RTDDTAP
"Press RETURN for real-time statistics for all DTAP messages

        |
        |
        |
```

To add the new application RTDNEW, insert the lines;

```
APP 2  RTDNEW
"Press RETURN for ........................."
```

2. The procedure section contains a procedure for every application in the Applications menu. The start of each procedure is defined by the name in the Applications menu.

   Example:
   (Lines starting with * are for comment only.)

```
*
*
RTDGSM
* Monitor the signaling link using the first real-time decode.
M

#CALL APPLIB_RTD1
In this example, the first real-time decode defined in the personality file is RTDGSM.
!

RTDGSM1
* Monitor the signaling link using the second real-time decode.
M
#CALL APPLIB_RTD2
!
                    |
                    |
                    |
#END
```

   To add the new application RTDNEW, insert the lines;

```
RDTNEW
* Comment describing the test.
M
#CALL APPLIB_RTDNEW
!
```

The Applications CALL Procedure

This procedure is called by the Applications File to control a self-contained test.

For example, if the active personality file specifies a real-time decode as the third available RTD (RTD3);

- `M` *Start monitoring*
- `R` *Select real-time decoding, (using the first defined decode)*
- `N` *Next (second) RTD*
- `N` *Next (third) RTD*

Written as;

```
MRNN
```

5

Test Sequences

In Emulation Mode, the HP 37900 acts as a device (for example a Signaling Point) in a signaling network. It can send signaling messages to other devices in the network. It can receive and respond to specified messages from the other devices.

A test sequence is a process which completely controls the actions of the HP 37900 during a test. For example;

- It includes INPUT statements to specify the received messages or other events it will accept and respond to.
- It includes OUTPUT statements to send messages onto the network.

It also controls wait periods, displayed messages, and so on. It can call in independent Pascal procedures to perform a wide range of actions.

Planning and Writing a Test Sequence

To create a test sequence you must first plan what activities you want the HP 37900 to perform. Draw a message sequence diagram, like the example below, to show the flow of messages.

A test sequence is a text file, written off-line in SDL (the Specification Description Language, as defined in CCITT Z.101). The HP 37900 uses a small subset of SDL.

You can either use the editor in the STS File Manager or quit the STS software and use the Pascal system Editor.

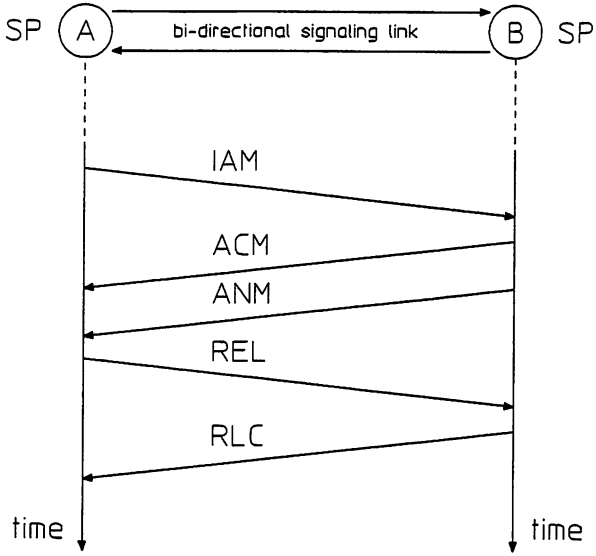
- In SDL, the complete test sequence (or program) is called a "process". It starts and finishes with `PROCESS` and `ENDPROCESS` statements.
- The process is divided into groups called "state-bodies". Each state-body starts with a STATE statement, and contains a number of INPUT statements each of which ends with a NEXTSTATE statement.
- Each state-body contains a number of action statements.

A process waits in a state until a valid input occurs. It then performs any required actions and changes to another state.

In the following example;

IAM = initial address message
ACM = address complete message
ANM = answer message
REL = release message
RLC = release complete message

This example shows a typical call setup sequence. The call is originated by Signaling Point A.


SP
A
bi-directional signaling link
B
SP
IAM
ACM
ANM
REL
RLC
time
time

```
PROCESS iam_test;

  SET IE_0;

  STATE iam;
    INPUT IE_0;
      OUTPUT IAM;
      OUTPUT 'IAM message sent';
      SET_TIMER 1000;
      NEXTSTATE w_f_acm;

    INPUT UNDEFINED;
      OUTPUT 'Input undefined ';
      NEXTSTATE iam;

  STATE w_f_acm;
    INPUT ACM;
      OUTPUT 'ACM reply received';
      RESET_TIMER;
      NEXTSTATE w_f_anm;

    INPUT TIMEOUT;
      OUTPUT 'Reply not received within 10s';
      STOP;

    INPUT UNDEFINED;
      OUTPUT 'Undefined input while waiting for ACM';
      NEXTSTATE w_f_acm;

      |

ENDPROCESS.
```

Figure 5-1. Example Process

Activities within a Process

A process waits in a state until a valid input occurs. It then performs a transition to another state or stops. During the transition it performs any specified actions, such as sending a message or displaying text.

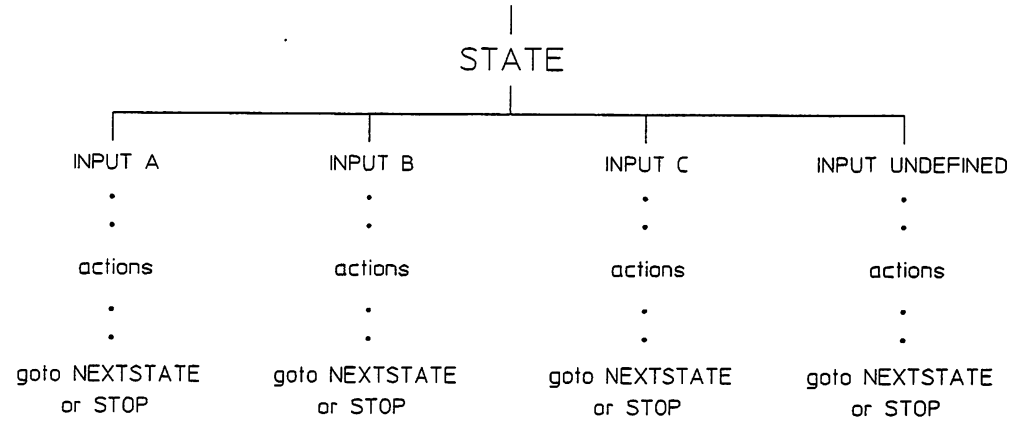


Figure 5-2. The State-body

In the diagram above, INPUT priority is from left to right. When an input event occurs, it is checked first against the template for INPUT A. In the following example a received message is compared first against the ACM message set up in the internal message catalog. If the input event matches INPUT A the action(s) specified immediately after INPUT A are performed.

If the input event does not match INPUT A it is compared with INPUT B and so on until a match is found. If it does not match any of the valid inputs it is called an UNDEFINED INPUT.

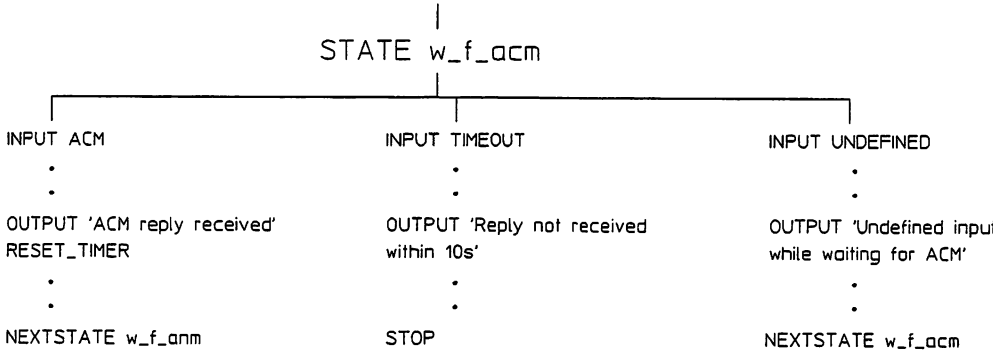


Figure 5-3. Example State-body

This example represents the STATE w_f_acm in the example process given earlier in this chapter.

5

Contents of a Process

This section gives a brief description of the main elements of a process and the required syntax.

A process contains the following elements in the order shown.

Element	Syntax
A heading statement giving the process name (mandatory).	`PROCESS <name>;`
Statements setting "internal event" flags or starting timers (optional).	`SET IE_<number>;` `SET_T<number> <timeout>;`

The first state-body (mandatory). The process always enters this state-body.	`STATE <name>;`
	`INPUT <NAME>;`
	Action statements
	`NEXTSTATE;` or `STOP;`

Subsequent state-bodies (maximum 5000).	
An end statement (mandatory).	`ENDPROCESS.`

All statements must be part of a state-body except for;

PROCESS <name>;
SET IE_<number>;
ENDPROCESS.

Punctuation

The following characters are required in the SDL syntax and must not be used in PROCESS or STATE names.

; must terminate every statement except `ENDPROCESS`.

. terminates the `ENDPROCESS.` statement.

' delimits a COMMENT or OUTPUT string. For example;

```
OUTPUT 'Starting test sequence';
```

You can use ; or . within the string delimiters ' '

The Heading Statement (PROCESS <name>)

Example;

```
PROCESS iam_test;
```

<name>
- Any number of alphanumeric and underscore characters, first character alphabetic. Upper or lower case.
- Only up to the first eight characters are used by the HP 37900 to identify the process, so they must be unique.
- Do not use statement words (CALL, INPUT, TIMEOUT and so on) or the following symbols in the name.
  Semi-colon ";"
  Full stop "."
  Inverted comma "'"

SET (Internal event Flags or Timers)

The SET statement is used to;

- Set internal event flags true.
- Start timers and set their timeout periods.

Internal events (IE_<number>)

<number> 256 flags available, IE_0 to IE_255.

An internal event is one of the valid input events. You can set an internal event flag (for example IE_0) so that a subsequent input statement (INPUT IE_0) accepts it as a valid input.

Example;

```
SET IE_0;
   |
   |
   |
INPUT IE_0;
```

5

Refer to Figure 5-1. A process must wait in a state until a valid input is detected. In the example, the flag IE_0 is used to provide an input to start the process.

NOTES:

- The input statement resets the internal event flag to false.
- The SET IE_ statement is the only action statement which can be used outside a state-body.

Timers (T<number> <period>)

<number> Sixteen timers, T0 to T15

<period> 0 to 16777215 centiseconds (maximum approximately 46.5 hours).

Example;

```
SET_T0 1000
```

This sets the timeout period for timer T0 to 10 seconds and starts it running.

A subsequent input statement (INPUT TIME_T0) waits for T0 to reach its timeout and responds to it as a valid input event.

NOTES:

- If a timer is set when it is already running, it is reset and starts again to run for the new period.
- When a timer reaches its timeout it resets to zero.
- `SET_TIMER` is accepted as a valid statement, equivalent to `SET_T0`.
- `CLR_T<number>` stops the timer and resets it to zero.
  `CLR_ALL;` stops and resets all timers.

5

The STATE and NEXTSTATE Statements

The STATE statement must be the first statement in the state-body. It assigns a statename to its state-body. A NEXTSTATE statement uses a statename to instruct the process where to go next.

Example;

```
STATE w_f_acm;
```

(wait for ACM)

Figure 5-3 shows how, for each different type of input, NEXTSTATE is used to direct the process to the next state-body. (In the INPUT UNDEFINED branch, `NEXTSTATE w_f_acm` means "remain in this state until a valid input is received".)

The INPUT Statement

The input statement instructs the process to watch for a specified event. When the event has occurred the process moves to the next statement in the statebody.

Example	Meaning
`INPUT ACM;`	Compare received message against the ACM template in the message catalog, if a match occurs read the next statement.
`INPUT IE_9;`	If internal event flag IE_9 has been set true, reset it and read the next statement.
`INPUT TIME_T1;`	If timer T1 has reached its timeout, read the next statement.
`INPUT UNDEFINED;`	If an input event occurs which does not match another INPUT statement in this state-body, read the next statement.

The OUTPUT Statement

5

An OUTPUT statement either sends a test message out on the signaling link, or a text message to the display.

Example	Meaning
`OUTPUT ACM;`	Send a copy of the message ACM held in the internal message catalog.
`OUTPUT 'ACM message sent';`	Display the text enclosed in single inverted commas.

The CALL Statement

The CALL statement temporarily transfers control of the HP 37900 from the test sequence to an independent user-written Pascal procedure, known as a CALL Procedure.

Typical uses of CALL are:

Real-time manipulation of run-time variables. (See "Use of Run-time Variables".)
Incrementing and decrementing counters.
Writing received messages to the display or test log.
Performing repetitive tasks.
Interactive test sequences with user prompts.
Automatic test procedure with flow control.
Manipulation of message contents either automatically or interactively.
Sending a series of similar messages.
Incrementing a telephone number automatically.
Counting errors and logging them.

Syntax;

```
CALL <'name'>;
```

The name is the name of the required procedure, and can include parameters to be passed to the procedure.

```
CALL 'mod_name_user_proc parm1 parm2';
```

Where;

`mod_name` is the name of the module where the CALL procedure is stored.

`user_proc` is the name of the CALL procedure.

`parm1` is the first parameter (if required).

`parm2` is the next parameter.

A set of procedures is provided with the product to help you write your own procedures. They are in the `call_lib` module. These are described under "The HP CALL Library" later in this chapter.

Example;

```
PROCEDURE log_user_string ( user_string : user_string_type ) ;
```

This places a user-defined string in the test log. (This procedure is used in the "CALL Procedure Example" later in this chapter.)

The CALL statement to run this procedure is;

```
CALL 'call_lib_log_user_string <string>';
```

Where;

`call_lib` is the name of the module where the CALL procedure is stored.

`log_user_string` is the name of the CALL procedure.

`<string>` this parameter is the string of text to be logged.

A sequence can use up to 200 Call Procedures.

A compiled version of each module containing CALL procedures (previously compiled using the Pascal Compiler) must be loaded before execution of the sequence, use
`L - Load SDL CALL procedures` in the Emulation Mode menu.

Only the syntax of the CALL statement is checked when the sequence is compiled. The presence of the procedures is not checked until you execute the sequence.

If you receive this error message during sequence compilation;
`CALL expected`
check the syntax of the CALL statement.

CALL Procedures

A set of procedures is provided with the product. This section describes how to use them in writing your own procedures. They include the examples below, and are listed and described under "The HP CALL Library" later in this chapter.

Use of Run-time Variables

When you define a signaling message, you can set up "run-time operations".

The SAVE operation saves the value of a specified part of the message to a run-time variable `var_name` when the message is transmitted or received. (You have to provide a name for `var_name`.)

The RESTORE operation assigns the value of run-time variable `var_name` to the specified part of the message when it is transmitted or received.

5

A CALL procedure can access `var_name` and use it in any Pascal operation.

Examples;

```
PROCEDURE restore_pos_int ( var_name : var_name_type ;
                            VAR pos_int : pos_int_type ;
                            VAR bit_total : bit_total_type ) ;

PROCEDURE save_pos_int ( pos_int : pos_int_type ;
                         bit_total : bit_total_type ;
                         var_name : var_name_type ) ;
```

These procedures are described under "The HP CALL Library".

In the following example, a message run-time operation has saved the value of its CIC to the run-time variable `'CIC'`. The procedure copies this value to the Pascal variable `pos_int`, increments it, and copies the new value back to `'CIC'`.

The next run-time restore operation will assign this value to its CIC.

CALL Procedure Example

```
MODULE user_lib;
```
Name the module in which all required procedures will be stored.

```
$SEARCH '*INTERFACE.', '*PAWSLIB.',
        '*STSLIB.CODE', '*EMU.CODE'$
```
State where to search for variables.

```
IMPORT SYSGLOBALS , sts_globals ,
                    call_lib;
```
Call in global variables for use.

```
EXPORT PROCEDURE increment_cic
   ( user_string : user_string_type );
```
Make the procedures accessible to other procedures.

```
IMPLEMENT
```
Begin block 1.

```
PROCEDURE increment_cic
  ( user_string : user_string_type );
```
Name the procedure.

```
VAR
   pos_int : pos_int_type ;
   bit_total : bit_total_type;
```
Declare local variables.

```
BEGIN
```

```
log_user_string
     ( 'increment_cic entered' );
```
Send text to test log.

```
restore_pos_int
     ( 'CIC' , pos_int , bit_total );
```
Put value of CIC into pos_int.

```
pos_int := pos_int + 1 ; save_pos_int
     ( pos_int , bit_total , 'CIC' );
```
Increment pos_int and save its new value into CIC.

```
log_user_string
     ( 'increment_cic exited' );
```
Send text to test log.

```
END;

END.
```

The HP CALL Library

A set of library procedures is provided with the product, to help you build your own procedures. They are stored in the `call_lib` module.

Some of these procedures have the same or similar functions as SDL statements, so they are not called from test sequences. Another procedure can use them and then continue to run. This avoids the need to end, return to the sequence, then start another procedure.

The heading statements of procedures called from test sequences must have the following format, even if the `<user_string>` parameter is not required.

```
PROCEDURE <procedure_name> ( user_string : user_string_type ) ;
```

Procedures Callable from a Test Sequence

5 These procedures can be called directly from a test sequence. They can also be called from another procedure.

Cancel Notification of Link Failures (CancelNotifyLinkFail)

```
PROCEDURE CancelNotifyLinkFail ( user_string : user_string_type ) ;
```

This procedure disables notification of link failures (see "Enable Notification of Link Failures (SetNotifyLinkFail)" for details).

The parameter <user string> is not required in the CALL statement.

Enable Notification of Link Failures (SetNotifyLinkFail)

```
PROCEDURE SetNotifyLinkFail ( user_string : user_string_type ) ;
```

This procedure enables notification of link failures. It takes one integer parameter *BaseIE*.

If an operational link goes out of service, Emulate informs the test sequence by setting internal event *BaseIE + link*. If an out-of-service link becomes operational, Emulate informs the test sequence by setting internal event *BaseIE + 8 + link*. It is then up to the test sequence to act on these internal events.

This procedure is useful when using the automatic realignment feature for No7 links, as the out-of-service message is not displayed.

Set Level 2 Error Count (SetL2ErrorCount)

```
PROCEDURE SetL2ErrorCount
  ( user_string : user_string_type );
```

This procedure sets the level 2 error counter to the integer value of the first parameter in `<user_string>`.

When running Emulation at test level 2, errored frames/SUs are returned with an explanation of the type of error. These messages are written to the test log as they occur. The level 2 error count is incremented each time a level 2 error occurs.

Check Level 2 Error Count (CheckL2ErrorCount)

```
PROCEDURE CheckL2ErrorCount
  ( user_string : user_string_type );
```

This procedure checks the current value of the level 2 error counter against the integer value of the first parameter in `user_string`, passed from the calling statement.

If the level 2 error counter is less than or equal to the value supplied internal event 8 is set, otherwise internal event 9 is set.

Log Level 2 Error Count (LogL2ErrorCount)

```
PROCEDURE LogL2ErrorCount
  ( user_string : user_string_type );
```

This procedure writes the current value of the level 2 error counter to the test log.

The parameter <user string> is not required in the CALL statement.

Disable Pass Through (DisablePassThru)

```
PROCEDURE DisablePassThru ( user_string  : user_string_type ) ;
```

This procedure disables pass-through mode. This allows the test sequence to choose when to start accepting messages. In particular, it allows pass-through mode to be enabled in the short period between *Starting* a link and *Executing* a test sequence, when input messages might otherwise be discarded.

5

The parameter <user string> is not required in the CALL statement.

Enable Pass-Through (EnablePassThru)

```
PROCEDURE EnablePassThru ( user_string  : user_string_type ) ;
```

This procedure enables pass-through mode. Call it before the test sequence exits if messages are to be passed through while the test sequence is not running.

Note

In Pass-through mode, a direct "signaling connection" is made between pairs of links. Signaling data passes through a signaling connection without being logged by the HP 37900.

The parameter <user string> is not required in the CALL statement.

Log User String (log_user_string)

```
PROCEDURE log_user_string ( user_string : user_string_type ) ;
```

This procedure writes **<user_string>** to the test log with type *CALL* and timestamps it. This feature provides a useful way of tracing the progress of a user-defined procedure.

The parameter <user string> is not required in the CALL statement.

Replace Keyboard [Stop] ISR (SetupStopISR)

Normally, when the Stop key is pressed, the keyboard Interrupt Service Routine (ISR) causes the test sequence to stop instantly. The `SetupStopISR` procedure is used to stop the sequence in a controlled way, for example to prevent messages being lost or to close files before stopping. It does this by replacing the keyboard ISR with one which intercepts the Stop key, preventing the sequence from stopping until the required tasks have been completed.

```
PROCEDURE SetupStopISR ( user_string : user_string_type ) ;
```

This procedure takes two parameters;

- Internal event number (for example `IE_100`)
- Re-enable pass-through flag (`Y` or `N`, default is No)
  This parameter is used for a specific application, as described below.

When the Stop key is detected, the specified internal event is set and, if selected, pass-through is re-enabled. (Subsequent Stop key presses cause the message `Test sequence stopping ...` to be displayed.) It is then up to the test sequence to detect the specified internal event and perform the required tasks before stopping.

The example below is a section of a sequence which receives messages on one link, processes them, and passes them out on another link. When the sequence calls `SetupStopISR`, the standard keyboard ISR is replaced with one which intercepts the [Stop] key.

If the Stop key is pressed, pass-through is re-enabled so that no more messages are logged and `IE_100` is added to the end of the message queue.

When the internal event is received by the test sequence, it knows that its queue is now empty. The test sequence must then re-instate the standard keyboard ISR by calling `ReinstateKeybdISR` and then terminate using a `STOP` statement.

Reinstate Keyboard [Stop] ISR (ReinstateKeybdISR)

```
PROCEDURE ReinstateKeybdISR ( user_string : user_string_type ) ;
```

This procedure re-instates the standard keyboard ISR (see "Replace Keyboard [Stop] ISR (SetupStopISR)" for more details).

The parameter <user string> is not required in the CALL statement.

Set ISDN Acknowledged Operation (SetISDNAckOp)

```
PROCEDURE SetISDNAckOp
  ( user_string : user_string_type );
```

This procedure causes subsequent ISDN level 3 messages to be sent in I frames. This means the messages will be acknowledged.

The parameter <user string> is not required in the CALL statement.

Set ISDN Unacknowledged Operation (SetISDNUnAckOp)

```
PROCEDURE SetISDNUnAckOp
  ( user_string : user_string_type );
```

This procedure causes subsequent ISDN level 3 messages to be sent in UI frames. This means the messages will not be acknowledged.

The parameter <user string> is not required in the CALL statement.

Set Test Log Size (SetTestLogSize)

```
PROCEDURE SetTestLogSize ( user_string : user_string_type ) ;
```

This procedure allows you to limit the size of the Emulate test log, for example to free heap space for use by user-defined CALL procedures or to make the binary version of the test log fit on a floppy disc.

This procedure takes one integer parameter `<no_items>` which specifies the size of the test log as the number of log items it holds (at least 2).

NOTE: Each log item uses 300 bytes of heap space.

```
PROCESS xxxx;

SET_IE 0;

STATE INIT;

  INPUT IE_0;
     CALL 'CALL_LIB_SetupStopISR 100 Y';
     CALL 'DisablePassThru';
     NEXTSTATE BODY;

STATE BODY;
     |
     |
     |
  INPUT IE_100;
     CALL 'CALL_LIB_ReinstateKeybdISR';
     STOP;

ENDPROCESS.
```

5

Note

Autorepeat mode (an option in the START LINKS menu) must NOT be on when using this facility.

Stop Control File (stop_control_file)

```
PROCEDURE stop_control_file ( user_string : user_string_type ) ;
```

This procedure stops the test sequence reading from a test control file, so that when the test sequence terminates control returns to the keyboard.

The parameter <user string> is not required in the CALL statement.

Take Links out of Service (TakeLinksOutOfService)

```
PROCEDURE TakeLinksOutOfService ( user_string  : user_string_type ) ;
```

This procedure takes all selected links out of service.

Before doing so it disables automatic Realignment (if enabled) to prevent Emulate from trying to realign the selected links.

The parameter <user string> is not required in the CALL statement.

Text to Remote (text_to_remote)

```
PROCEDURE text_to_remote ( user_string : user_string_type ) ;
```

This procedure sends <user_string> to the remote host, if it is logged in.

Procedures Not Callable from a Test Sequence

5

The following procedures are called from other procedures, they can not be called from test sequences.

Extract Integer (ExtractInteger)

```
PROCEDURE ExtractInteger
  (     input_string : user_string_type;
    VAR startpos     : integer;
        min_value    : integer;
        max_value    : integer;
    VAR int_value    : integer;
    VAR int_ok       : boolean          );
```

This procedure extracts an integer <int_value> from <input_string> starting at position startpos and ensures it lies within the range min_value .. max_value. If it is not a valid integer or is out of the specified range, int_ok is set false.

On exit, startpos points at the next character to be read.

This procedure is useful for extracting integer parameters from the CALL parameter string.

Get Message Contents (get_msg_contents)

```
PROCEDURE get_msg_contents (     msg_name : MessageNameType ;
                             VAR message  : message_type ) ;
```

This procedure copies the contents of message <msg_name> from the internal message catalog into message.

Log User Message (log_user_message)

```
PROCEDURE log_user_message ( message     : message_type ;
                             link_number : emu_link_type ) ;
```

This procedure inserts the user-defined message message into the test log. This simulates a message arriving on a particular link.

The test sequence then processes this message in the same way as a message arriving on link <link_number>, except that it is logged with type *CALL*.

5 NOTE: To avoid catastrophic failure of Emulation, the first three bytes of the message must conform to the following format, which defines the length of the message (including byte 0):

byte -2 must contain the *least significant* byte of the message length, that is, length MOD 256.

byte -1 must contain the *most significant* byte of the message length, that is, length DIV 256. Therefore the length of the message (including byte 0) is;

```
(message[-1] *256) + message[-2]
```

byte 0 If the length of the message is less than or equal to 62, this must be set to the length of the message. If the length is greater than 62, it must be set to 63.

Reset SDL Timer (reset_sdl_timer)

```
PROCEDURE reset_sdl_timer ( sdl_timer_num : sdl_timer_num_type ) ;
```

This procedure resets the SDL timer <sdl_timer_num>.

This provides the same functionality as the *CLR_T<number>* statement in SDL.

Restore Positive Integer (restore_pos_int)

```
PROCEDURE restore_pos_int (     var_name  : var_name_type ;
                            VAR pos_int   : pos_int_type ;
                            VAR bit_total : bit_total_type ) ;
```

This procedure reads the contents of the run-time variable <var_name> into <pos_int> and returns the number of bits restored in <bit_total>.

NOTE: The maximum size of a positive integer is 31 bits.

Restore Run-Time Data (restore_rt_data)

```
PROCEDURE restore_rt_data (        var_name     : var_name_type ;
                           ANYVAR message      : message_type ;
                                  last_octet   : octet_number_type ;
                                  octet_number : octet_number_type ;
                                  bit_number   : bit_number_type ;
                             VAR bit_total     : bit_total_type ) ;
```

This procedure reads the contents of the run-time variable <var_name> into message.

<octet_number> and <bit_number> — specify where in the message the data is placed.

<last_octet> — specifies the maximum size of the data structure and allows the procedure to perform some basic checks while copying the data.

<bit_total> — specifies the total number of bits restored.

NOTE: ANYVAR allows you to use a more convenient data structure, but **this feature must be used with great care.**

For example you can write to a smaller data structure, but you must take care not to try to write past the end of the variable.

Save Positive Integer (save_pos_int)

```
PROCEDURE save_pos_int ( pos_int   : pos_int_type ;
                         bit_total : bit_total_type ;
                         var_name  : var_name_type ) ;
```

This procedure saves the contents of <pos_int> into the run-time variable. <bit_total> sets the maximum size of the variable.

NOTE: The maximum size of a positive integer is 31 bits.

Save Run-Time Data (save_rt_data)

```
PROCEDURE save_rt_data ( ANYVAR message      : message_type ;
                                 last_octet   : octet_number_type ;
                                 octet_number : octet_number_type ;
                                 bit_number   : bit_number_type ;
                                 bit_total    : bit_total_type ;
                                 var_name     : var_name_type ) ;
```

This procedure saves the contents of <message> into the run-time variable <var_name>.
<octet_number>, <bit_number> and <bit_total> identify the information to be stored.

<last_octet> specifies the maximum size of the ANYVAR data structure and enables the procedure to make basic checks on the integrity of the data.

See "Restore Run-time Data (restore_rt_data)" for a note on ANYVAR.

Send Message (send_message)

```
PROCEDURE send_message ( message     : message_type ;
                         link_number : emu_link_type ) ;
```

This procedure sends a specified message on a specified link. This has the same function as the *OUTPUT* statement in SDL.

NOTE: The first three bytes of the message must conform to the following format, which defines the length of the message (including byte 0):

byte -2 must contain the *least significant* byte of the message length, that is, length MOD 256.

byte -1 must contain the *most significant* byte of the message length, that is, length DIV 256. Therefore the length of the message (including byte 0) is;

```
(message[-1] *256) + message[-2]
```

byte 0 If the length of the message is less than or equal to 62, this must be set to the length of the message. If the length is greater than 62, it must be set to 63.

5

FlushLastFrame (user_string : user_string_type)

```
PROCEDURE FlushLastFrame ( user_string :user_string_type );
```

This procedure instructs the specified link to send a copy of the last message upto the SDL program which can then be captured using an appropriate INPUT command.

It takes as a parameter, the link number in the range 1 - 8.

Set IE <number> (set_internal_event)

```
PROCEDURE set_internal_event
          ( internal_event : internal_event_type ) ;
```

This procedure sets internal event number <internal_event> true. This has the same function as the *SET_IE* statement in SDL.

Set SDL Timer <number> (set_sdl_timer)

```
PROCEDURE set_sdl_timer
          ( sdl_timer_num   : sdl_timer_num_type ;
            sdl_timer_delay : sdl_timer_delay_type ) ;
```

This procedure sets the SDL timer <sdl_timer_num> to <sdl_timer_delay>, where the delay is specified in hundredths of a second. This has the same function as the *SET_T<number>* statement in SDL.

5

Programming Notes

Try..Recover Escapecodes

Escape codes numbered up to 1000 are reserved for HP 37900 use. Use escape codes greater than 1000 in user-procedures.

The escapecodes that the CALL library generates are trapped at the outermost level of test sequence execution and an explanatory message is displayed. The test sequence stops when an escapecode is detected.

Use `ESCAPE <escapecode>` to return escapecodes less than 1000 to the Emulation software for processing. For further information, refer to the *HP Pascal Language Reference* (Part Number 98615-90053) under "System Programming Language Extensions".

Memory Space

The global memory space allocated for user-procedures within Emulation is limited to 200 bytes. Reduce use of this space by permanently loading into memory files containing declarations for global variables and types, and those containing user-procedures which do not use the Call library.

Exit the HP 37900 Main Menu, and use the Permanent command from the Pascal command line. Use the IMPORT command to link permanently loaded and Emulation loaded files.

Timers and Counters

In most cases, user-procedures cannot disrupt the operation of the HP 37900. However, they can affect system timers.
For example, Emulation treats an incoming message as an interrupt, and while dealing with this, disables all keyboard and timer interrupts.

There are also timer interrupt handlers for the following Pascal system timers: DELAY, MATCH and CYCLIC (when under remote control).

Run-time Variables

Maximum Size

The first save operation to a run-time variable determines its maximum size. If your sequence uses a variable more than once, start by ensuring it is big enough for all operations.

For example, the function of a sequence might be to receive all messages, modify some of them and pass all messages on. The easiest way is by using run-time variables. To allow different lengths of message to be saved to the same variable, use a CALL procedure to initialise the variable to its maximum size at the start of the sequence.

Example;

```
PROCEDURE InitRTVars ( user_string : user_string_type ) ;

VAR
  dummy : message_type ;

BEGIN

  save_rt_data (dummy, max_octet_number,
                1, 1, 8 * max_octet_number, 'PASS');

END ;
```

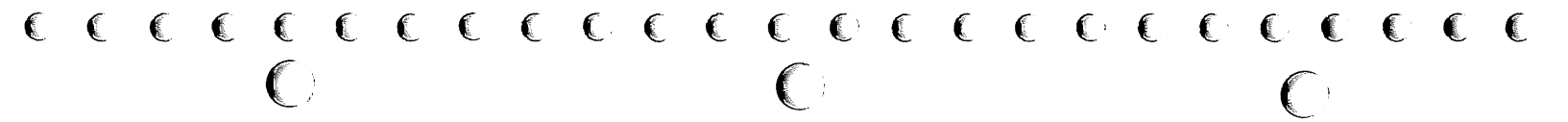
Avoiding Accidental Overwriting

The value of a run-time variable only exists until the next <save> operation is performed on it. When a *save* operation is performed to a run-time variable, use the new value as soon as possible afterwards.

For example;

```
INPUT msgx ;
  COMMENT 'msgx saves octet 1, bit 1, 8 bits to variable OCTET1';
    |
    |
```

Use the new value of OCTET1 in the same state-body, before another incoming message which saves to OCTET1 can be received. Do this either by CALLing a procedure or OUTPUTting a message which restores OCTET1.



6

Message Input Files

You can create a message by writing a message Input File in advance. When you are setting up the message catalog to be used by a sequence, you can read the input file instead of defining each octet individually. At the start of the file, define whether you are using binary, hexadecimal or decimal.

`$ BIN_ON $` switches into binary mode (example 00000000 11110000 01010101)
`$ HEX_ON $` switches into hex mode (example A3 FF 34 1 2 3)
`$ DEC_ON $` switches into decimal mode (example 10 255 3 4). This is the default.
`$ END $` indicates end of message, all input following is ignored.

Any text following a "#" is ignored, and blank lines are ignored.

Example;

```
$ HEX_ON $
11
22
33
44
55
$ END $
```

When you read in this file the following hexadecimal values are allocated to the specified octets.

Octet 1 - 11
Octet 2 - 22
Octet 3 - 33
Octet 4 - 44
Octet 5 - 55

7

Message Decodes

Part 1 - HP-Supplied Decodes

The table following shows the list of standards supported by the HP 37900 and the associated decodes.

The first three columns describe the decode, the last column gives the name assigned to it by Hewlett-Packard.

The names comply with the convention illustrated in the following examples.

CRTUP C denotes CCITT.

R denotes Red Book version.

TUP - Telephone User Part.

A8ISUP A denotes ANSI.

8 denotes 1988 version.

ISUP - ISDN User Part.

Table 7-1. HP-Supplied Decodes

Standard	Version	Decode	Name
Q.703	Red Book	Q.703	CRQ703
Q.703	Blue Book	Q.703	CBQ703
Q.704	Red Book	MTP	CRMTP
Q.704	Red Book	SNM	CRSNM
Q.707	Red Book	MTN	CRMTN
Q.721-Q.724	Red Book	TUP	CRTUP
Q.761-Q.764	Red Book	ISUP	CRISUP
Q.711-Q.714	Red Book	SCCP	CRSCCP
Q.704	Blue Book	MTP	CBMTP
Q.704	Blue Book	SNM	CBSNM
Q.707	Blue Book	MTN	CBMTN
Q.721-Q.724	Blue Book	TUP	CBTUP
Q.761-Q.764	Blue Book	ISUP	CBISUP
Q.711-Q.714	Blue Book	SCCP	CBSCCP
Q.711-Q.714	Blue Book	SCMG	CBSCMG
Q.771-Q.775	Blue Book	TCAP	CBTCAP
*Q.921	Blue Book	Q.921	CBQ921
*Q.931	Blue Book	Q.931	CBQ931
Q.704	White Book	MTP	CWMTP
Q.704	White Book	SNM	CWSNM
Q.761-Q.764	White Book	ISUP	CWISUP
Q.711-Q.714	White Book	SCCP	CWSCCP
Q.711-Q.714	White Book	SCMG	CWSCMG
Q.771-Q.775	White Book	TCAP	CWTCAP
BTNR166	Issue 1	NUP	B1NUP
BTNR	Issue 1	SNM	B1SNM

Table 7-1. HP-Supplied Decodes (continued)

Standard	Version	Decode	Name
GSM23	1989 (04.08) 3.3.1	DTAP	CBBSAP
GSM23	1989 (08.08) 3.7.0	BSSMAP	CBBSAP
GSM23	1989 (09.02) 3.2.1	MAP	CBMAP
GSM27	1990 (04.08) 3.9.0	DTAP	CBBSAP
GSM27	1990 (08.08) 3.9.2	BSSMAP	CBBSAP
GSM27	1990 (09.02) 3.6.0	MAP	CBMAP
SMG10	(08.08) 4.6.0	BSSMAP	BSAP10
SMG10	(04.08) 4.7.0	DTAP	BSAP10
SMG10	(04.11) 4.6.0	SMS	N/A
SMG10	(03.40) 4.6.0	SMS	N/A
SMG10	(08.58) 4.5.0	Abis	ABIS10
SMG10	(09.02) 4.6.0	MAP	MAP10
T1.111	ANSI 1986	MTP	A6MTP
T1.111	ANSI 1986	SNM	A6SNM
T1.111	ANSI 1986	MTN	A6MTN
T1.113	ANSI 1985	ISUP	A5ISUP
T1.113	ANSI 1988	ISUP	A8ISUP
T1.112	ANSI 1988	SCCP	A8SCCP
T1.112	ANSI 1988	SCMG	A8SCMG
T1.114	ANSI 1988	TCAP	A8TCAP
TA-TSV-954 Issue 1	Bellcore Mar89	ABS (LIDB)	B9LIDB
TR-NPL-246 Issue 1 Rev 2	Bellcore Jun 87	ISUP	B7ISUP
TR-NPL-246 Issue 1 Rev 2	Bellcore Jun 87	SCCP	B7SCCP
TR-NPL-246 Issue 1 Rev 2	Bellcore Jun 87	TCAP	B7TCAP
TR-NPL-246 Issue 1 Rev 2	Bellcore Jun 87	SNM	B7SNM
TR-NPL-246 Issue 1 Rev 2	Bellcore Jun 87	MTN	B7MTN
TR-NPL-246 Issue 1 Rev 2	Bellcore Jun 87	MTP	B7MTP
TR-NPL-246 Issue 1 Rev 2	Bellcore Jun 87	SCMG	B7SCMG

Table 7-1. HP-Supplied Decodes (continued)

Standard	Version	Decode	Name
TR-NWT-000533	Bellcore Jan 94	800 Series	B4800
TR-NWT-000954	Bellcore Nov 92	LIDB	B2LIDB
TR-NWT-000227	Bellcore Mar 93	CLASS	B3CLAS
TR-NWT-000220	Bellcore Mar 93	CLASS	B3CLAS
TR-NWT-000215	Bellcore Mar 93	CLASS	B3CLAS
TR-TSY-31 Issue 2	Bellcore Jun 88	Calling Number Delivery	B8CLAS
TR-TSY-32 Issue 1	Bellcore Nov 86	Bulk Calling Line Identification	B8CLAS
TA-TSY-215 Issue 1	Bellcore Sep 88	Automatic Callback	B8CLAS
TR-TSY-216 Issue 2	Bellcore Nov 88	Customer Originated Trace	B8CLAS
TR-TSY-217 Issue 2	Bellcore Nov 88	Selective Call Forwarding	B8CLAS
TR-TSY-218 Issue 2	Bellcore Nov 88	Selective Call Rejection	B8CLAS
TR-TSY-219 Issue 2	Bellcore Nov 88	Distinctive Ringing/Call Waiting	B8CLAS
TR-TSY-220 Issue 1	Bellcore Apr 89	Screening List Editing	B8CLAS
TR-TSY-227 Issue 1	Bellcore Sep 88	Automatic Recall	B8CLAS
TR-TSY-391 Issue 2	Bellcore Jun 88	Calling Number Delivery Blocking	B8CLAS
TR-TSY-533 Issue 2	Bellcore Jly 88	800 Services	B7800
TR-45.4	Dec 93 version 1.0.3	CDMA	CDMA
National Common Channel Signaling System	Helsinki 1988	TUP	F8TUP
FTZ 1TR7	1987	SCCP	G7SCCP
FTZ 1TR7	1987	ISUP	G7ISUP
FTZ 1TR7	1987	MTP	G7MTP
Nordic HUP	4th Edition	NMT-900	N9HUP
Nordic MUP	Rev. 2	NMT-900	N9MUP
Nordic MTP	4th Edition	NMT-900	N9MTP
CTC	Chile Jul 91	(CTC)ISUP	CTISUP

Part 2 - User defined Decodes

This chapter provides the detailed information you need to write your own decodes.

It is assumed that you know:

- How to use the Pascal Editor and Filer programs to manipulate, edit and compile text files.
- How to use personality files and decodes to ensure correct decoding of a signaling protocol.

The basic steps in writing your own decode are:

1. Write your decode in PASCAL, using HP-supplied library procedures for screen formatting.
2. Compile your decode to produce a "segment", which can be called by the HP 37900 software.
3. Register your decode in a personality file. (The HP 37900 decode-control software ascertains from the personality file which decode to use.)

Example Decode Sequence

A message is decoded in up to three parts. These are the Message Transfer Part (MTP), the Service Part, and (if required) the UserData part. A user-defined decode can be used to replace any of these parts. This example shows how these decodes combine to produce a complete message decode.

Consider decoding an SCCP CREF message, containing an ISUP message in its Data parameter. The CREF message is defined in the CCITT Red Book, Q.713 section 4.4. It will help you to understand the following discussion if you first read this section of the Red Book.

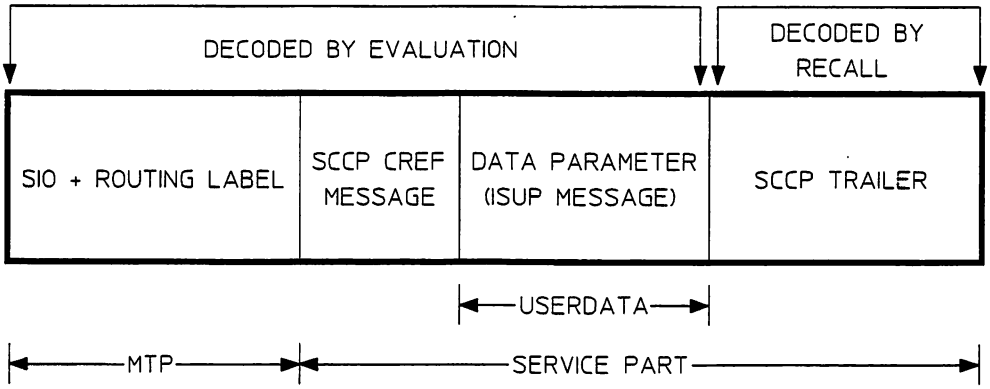


Figure 7-1. SCCP Structure

7 This is one of the more complex messages. As well as the EVALUATION procedure, which decodes up to and including the UserData Part, it also requires the RECALL procedure to decode the SCCP trailer.

Assume Personality 1 is CCITT Redbook, and that the personality file includes the following lines :

MP 1 *CRMTP*

SV 1 3 XXXX *CRSCCP*

UD 1 3 *CRISUP*

General Description of the Sequence

In the personality file above;

1. The first line shows that Personality 1 assigns the decode *CRMTP* to decode the Message Transfer Part (the SIO and Routing Label).

   MP — shows that the decode being assigned in this step is for the Message Transfer Part (MTP).

   1 — identifies which personality (1 or 2) in the active personality file assigns the decode *CRMTP*.

   CRMTP — is the name of the assigned decode.
   C denotes CCITT.
   R denotes Red Book version.
   MTP Message Transfer Part.

In this example, when CRMTP decodes the SIO, the SI value is decoded as 3.

2. The second line shows that Personality 1 assigns *CRSCCP* to decode the Service Part of a message whose SIO value is 3.

   SV — shows that the decode being assigned in this step is for the Service Part (otherwise known as the User Part).

   1 — identifies the personality.

   3 — When assigning the Service Part decode, the number in this column denotes the SI value.

   XXXX — This means that the Sub-Service Field (SSF) value does not matter and the decode is determined by the SI value alone.

   CRSCCP — is the name of the assigned decode (see the explanation for *CRMTP* above).

   Note that up to two alternative decodes could be assigned.

In this example, when CRSCCP decodes the Service Part, the Sub-Service Number (SSN) is decoded as 3. This indicates that the SCCP contains an ISUP message.

3. The third line shows that Personality 1 assigns *CRISUP* to decode the UserData Part of a message whose SSN value is 3.

UD	shows that the decode being assigned in this step is for the UserData Part.
1	identifies the personality.
3	When assigning the UserData Part decode, the number in this column denotes the Sub-Service Number (SSN), which identifies the user function (ISUP in this example).
CRISUP	is the name of the assigned decode. Note that up to two alternative decodes could be assigned.

4. When the UserData has been decoded, it is still necessary to decode the SCCP trailer. The *SCCP* decode is recalled (using the RECALL procedure, as described later).

Detailed Description of the Sequence

When a decode has extracted the information from the part of the message it is decoding, it passes information to the next decode via a number of **global** variables, identified by names beginning with "g_".

A simplified picture of the interactions is shown on the next page.

1. The MTP decode segment *CRMTP* is called by the HP 37900 decode software, with the 'position-in-message' variable **g_octet_number** set to 1 (the start of the message).

   This decode builds up the binary display, comment line and field decode information for the MTP section of the message.

   It returns **g_octet_number** set to the next octet to be decoded. It also returns the Service Indicator (SI) and SubService Field (SSF) values, which it extracted from the message, in **g_service_indicator** and **g_subservice_field**.

2. The values in **g_service_indicator** and **g_subservice_field** are used to select the Service decode from the SV lines in the current Personality (in this example it will be *CRSCCP*).

   This decode builds up the decode information for the octets from **g_octet_number** up to the length indicator of the Data Parameter. It returns:

   a. **g_octet_number** set to the NEXT octet to be decoded, which is the first octet of the Data parameter (see Figure 7-1).

   b. **g_user_data_present** set to TRUE.

   c. **g_user_data_SSN** set to the Sub-system number (SSN), which it extracted from the Called Party Address parameter.

   d. **g_recall** set TRUE.

3. The value in **g_user_data_SSN** is used to select the decode (in this example *CRISUP*) for the UserData part of the message. This decode builds up the decode information for the octets contained in the Data Parameter.

   It returns **g_octet_number** set to the NEXT octet to be decoded.

4. The Service decode *CRSCCP* is called again, via its RECALL entry-point, to process the SCCP trailer. (Normally this is just the 'End of optional parameters' marker.)

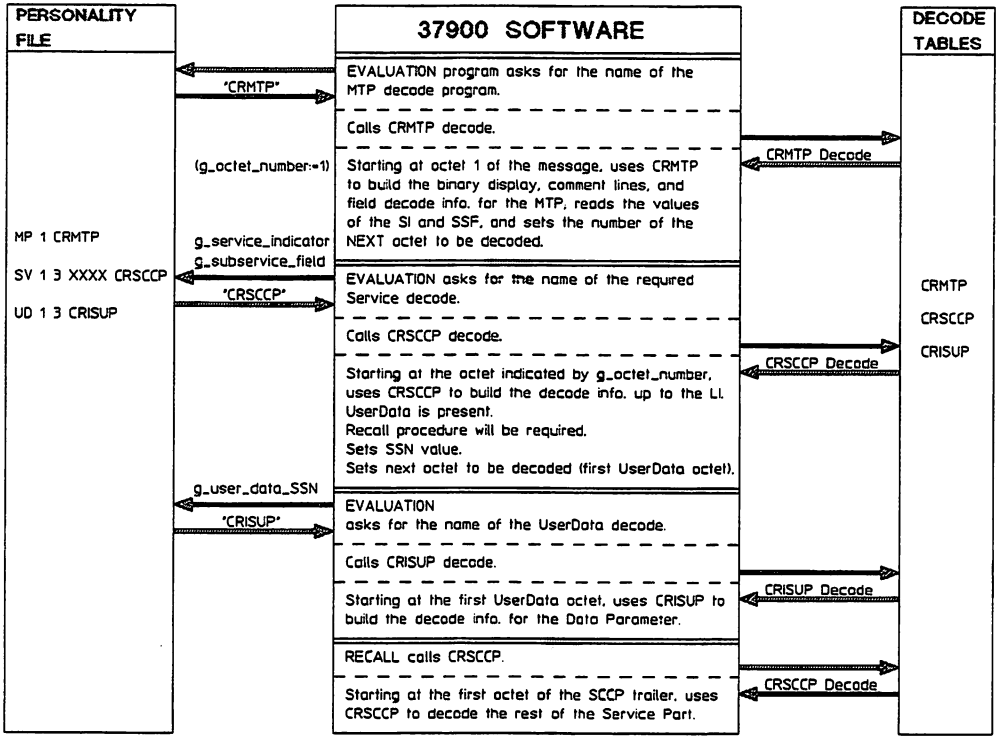


Figure 7-2. Decode Sequence

Naming Conventions

To be called by the HP 37900 decode control software, a user-defined decode **MUST** comply with the following naming conventions:

Decode Name: <decode_name>

The Decode Name is entered in the MP, SV, UD or KS line of the Personality file, to specify when the decode is to be used.
(KS is for Keystroke Selected decodes. During decoding of a message, the operator can override the assigned decode by pressing a key to select a keystroke-selected decode.)

All the names described below are derived from the Decode Name. It is represented by <decode_name> in their definitions.

The Decode Name can be up to 6 characters. The 6 characters MUST be allocated as follows :

character 1: U (for User defined).

character 2: to indicate the revision, typically this would be a single digit number.

characters 3-6: a mnemonic representing the protocol being decoded.

Example: <decode_name> = U3SCCP

Evaluation Procedure Name: <decode_name>EVALUATION

Example - U3SCCPEVALUATION

This procedure is called by the HP 37900 decode control software to build the decode up to, and including, the UserData part.

It is one of the two procedures EXPORTed by the user-defined segment.

Recall Procedure Name: <decode_name>RECALL

Example - U3SCCPRECALL

This procedure is called by the HP 37900 decode control software to build the trailer part of a message which contains UserData.

The RECALL procedure will only be called if the EVALUATION procedure exits with the **g_recall** variable set TRUE. If the EVALUATION procedure never does

this, then no RECALL procedure need be supplied.

Module name: M<decode_name>

The module containing the decode procedures MUST be called M<decode_name>.

Example - MU3SCCP

Segment name: <name>.SEG

The output from compilation of the decode module MUST be directed into a file called SYS:<decode_name>.SEG.

Example - SYS:U3SCCP.SEG.

(You must add the period as the final character to prevent the control software from adding the .CODE extension. If you fail to do so, the segment will not be found when the decode is called.)

Note

All of the names described above MUST use UPPER CASE characters.

Building a Decode

A template file for a user-defined decode, and an example decode, are included with the HP 37900 software.

The template provides the framework for building a user-defined decode. Its filename is **STSS:UDDTPLATE.TEXT**.

The filename of the example decode is **STSS:UDDXAMPLE.TEXT**.

Note

A user-defined decode passes information by reading and writing a number of **global** variables, identified by names beginning with "**g_**". These variables are listed and explained later in this chapter.

- These variables are already declared. They MUST NOT be declared again.
- They MUST ONLY be used in the way described here.

Working Example

If you are not yet familiar with creating or modifying decodes or personality files, study the previous chapter, "DECODES: Tutorial" and work through the examples to gain confidence.

Using Global Variables to Build the Message Decode

All the information to be displayed on the message decode screen is assembled by the EVALUATION procedure **<decode_name>EVALUATION**. This procedure MUST do the following:

g_octet_number is automatically assigned the number of the first octet to be decoded. The EVALUATION procedure must increment **g_octet_number** as each subsequent octet is decoded in turn until:

- The end of the message is reached.
- An unrecoverable error in the message format is detected. At this time the user-defined decode MUST exit.
- The end of *this part* of the message (for example the MTP part) is reached.

There are specific entry and exit values defined for each message part. These are detailed later in this chapter.

7

g_octet_info

The following figure is an extract from a message decode display. The information displayed for each octet (each individual line) is built by **g_octet_info**.

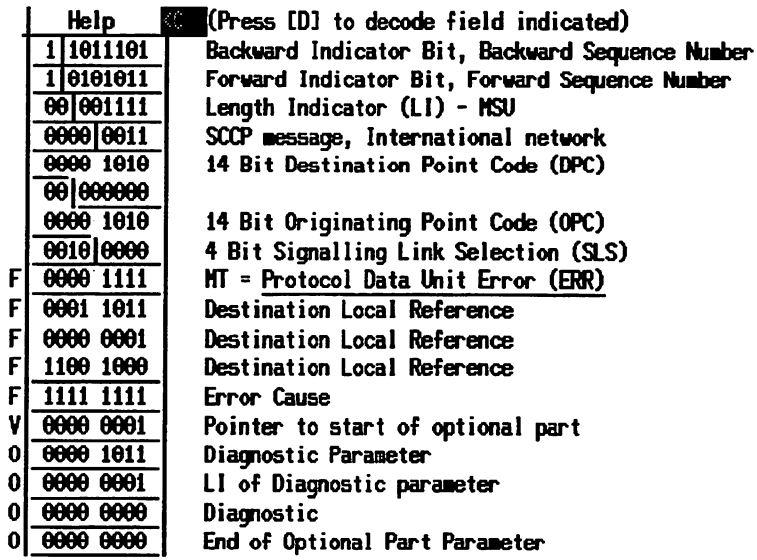


Figure 7-3.

Consider, for example, octet 9. It shows that each line consists of three sectors;

- The character in the first column, referred to as PART, (in this case the value is U).
- The binary string, referred to as STR.
- The comment text, referred to as COMMENT.

The three sectors are built by assigning values to the corresponding variables, which are all part of **g_octet_info**.

PART is the value given to **g_octet_info.part**
STR is the value given to **g_octet_info.octet_str**
COMMENT is the value given to **g_octet_info.comment**

The two other variables which complete **g_octet_info** are needed for the *field* decode of the octet. These are;

g_octet_info.proc - the name of the procedure which will give the field decode for the octet.

g_octet_info.anchor - the number of the first octet in the field being decoded.

PART	STR	COMMENT	PROCEDURE	ANCHOR
_info.part	**_info.octet_str**	**_info.comment**	**_info.proc**	**_info.anchor**

All of these variables must be defined for every octet to be decoded, even if in some cases the required value is " " (blank).

Building g_octet_info

g_octet_info.part

Assign a single character (which can be a space) to be displayed in column 1 of the screen line for the octet. This is used in decodes such as ISUP and SCCP using `F`, `V` and `O` to denote the Fixed, Variable and Optional parts of a message.

This column can be used for any purpose. It is recommended to enter 'U' to indicate that a user-defined decode is being used.

g_octet_info.octet_str

This is a 15-character array which contains the characters to be displayed in columns 2 - 14 of the screen line for the octet, and is the display of the octet in binary.

Thirteen characters represent the value read from the octet, the other two control underlining and have to be defined.

Do NOT write directly to **g_octet_info.octet_str**. You MUST use the supplied library procedures described under "Setting Up **g_octet_info.octet_str**" below.

For example, the procedure **FormatOctet** formats the value of the octet into the standard layout:

```
|bbbb bbbb|
```

g_octet_info.comment

Assign up to 63 characters for the comment line, to be displayed in columns 18-80 of the screen line for the octet.

g_octet_info.proc

Assign the name of a procedure which is to be called to give a field decode of the octet.

The named procedure must either be included in the user-defined decode, or be one of the supplied procedures described below under "DLIBRY Procedures".

The procedure writes to the 'decode area' of the screen (columns 18-80), using the procedures described below under "DA"'WRITE Procedures". ALL field decode writes to the screen must be done using these procedures.

If no field decode is needed. If no further decode is needed for an octet, (or if you have not yet written the field decode) assign the procedure **dlCannotDecode**. This is described under "DLIBRY Procedures".

g_octet_info.anchor

Assign the number of the first octet of the field being decoded.

For single octet fields, set this to **g_octet_number** (which is just the number of the octet being decoded).

For multiple octet fields, assign the number of the first octet of the field. The field decode procedure (named in g_octet_info.proc) will then be the same for each octet in that field.

g_octet_info.segment

The value in this field is set by the decode control software, and MUST NOT BE ALTERED.

g_octet_decode_info

The decode builds up the information for a given octet in **g_octet_info**. When it is complete, it is added to **g_octet_decode_info**. That is,

g_octet_decode_info^[g_octet_number] := g_octet_info ; (remember the ^)

g_octet_number is then incremented by 1,

g_octet_number := g_octet_number + 1 ;

Final Contents of g_octet_decode_info

g_octet_info[g_octet_number]
g_octet_info[g_octet_number+1]
g_octet_info[g_octet_number+2]
g_octet_info[g_octet_number+3]
g_octet_info[g_octet_number+n]

Note

- It is important to use **g_octet_info** because the correct value of the segment field is preset in this variable.
- When decoding multiple octet fields, set **g_octet_info.anchor = g_octet_number** on the first octet, then do not alter it for the remainder of the field.
- Having set **g_octet_info.part** to an appropriate character, there is no need to assign it on subsequent octets, unless it is to change.

Setting Up g_octet_info.octet_str

The standard layout for columns 2 - 14 is:

```
| bbbb  bbbb |
  8765  4321
```

(where b is a binary digit, 0 or 1). The layout can be enhanced by splitting into two fields and/or underlining. The bits are numbered as shown.

The octet in **g_message[g_octet_number]** is translated into display format in **g_octet_info.octet_str** using the procedures listed below. In general, up to 3 procedures can be called, and they must be in the following order.

1. **FormatOctet** translates the octet into the standard layout shown above.
2. **SplitFields (position)**, places a separator between bits **position** and **position-1**. Bits are numbered from 8 (most significant) to 1 (least significant). For example, **SplitFields (6)** gives the layout;

```
| bbb | bbbbb |
```

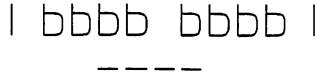
3. Use one ONLY of the following underlining procedures :
   a. **UnderlineField (LhsField)** or **UnderlineField (RhsField)**. This procedure can only be called if **SplitFields** has been called. It causes the selected side of the separator to be underlined. Following the above example, **UnderlineField (RhsField)** would gives;

```
| bbb | bbbbb |
        -----
```

b. **UnderlineSeparator.** This procedure can only be called if **SplitFields** has been called. It causes the separator to be underlined.
SplitFields (5);
UnderlineSeparator;
gives;

c. **UnderlineLeftBits (number_of_bits)** underlines the leftmost number_of_bits bits. **UnderlineLeftBits (5)** would result in;

d. **UnderlineBits (from_bit_number, number_of_bits)** underlines bits number_of_bits from from_bit_number to the right. **UnderlineBits (6,3)** gives;



e. **UnderlineOctet** underlines all of the bits (and the surrounding spaces).

```
| bbbb  bbbb |
------ - -----
```

4. **Special Case**

   Some specialised procedures exist for building up the display of Binary Coded Decimal strings, for example, as used for address digits in some CCITT #7 protocols. For each octet except the last in the BCD field, call procedure

   FormatBcd. For the last octet, call **FormatLastBcd.** This will produce the layout;

```
| bbbb | bbbb |
| bbbb | bbbb |
| bbbb | bbbb |
------ - -----
```

Note

If **FormatBCD** or **FormatLastBCD** has been called, no other calls (for example, **FormatOctet**, **SplitFields** or **Underline**) must be made for such octets.

7

Alternative Start Decode (SD)

Display of a message decode normally starts at the beginning of the message. The display can, however, be made to start at the beginning of the area of interest. The SD parameter has to be set in the personality file. In the decode, **g_start_display** defines the octet at which the display starts, see "Entry/Exit Conditions ... ".

Remaining Undecoded Octets

The decode control procedure, which is part of the standard HP 37900 software, will handle any undecoded octets.

Error conditions

If a Pascal error is encountered during decoding, the user-defined decode will exit. The standard HP 37900 decode control software will display an error message, and a prompt to press Return to continue.

When you press Return, the decode screen is displayed. This will show the binary values of all octets in the message, and the successful decode results up to the point where the decode failed. The octet at which the decode failed is indicated by a message. The remaining octets are labeled `Undefined octet`.

Entry/Exit Conditions for MTP Part Decodes

On Entry:

- **g_recall** = false
- **g_cic** = **undefined_cic**
- **g_message** contains the message to be decoded.
- **g_message_length** contains the length of the message in octets.
- **g_octet_number** is the index within **g_message** of the first octet of the portion of the message which the called procedure should decode.
- **g_octet_info.segment** contains a code which must be set into **g_octet_decode_info**^[x].**segment**, to allow the correct segment to be loaded when **g_octet_decode_info**^[x].**proc** is called.

More than one of these segments will be used in decoding a message.

On Exit:

- **g_octet_decode_info** should have been set up as far as **g_octet_number-1**.
- **g_octet_number** is the index within **g_message** of the next octet to be decoded.

- **g_service_indicator** contains the Service Indicator (SI) field extracted from the message.
- **g_subservice_field** contains the Sub-Service Field (SSF) field extracted from the message.
- **g_recall** = TRUE if the decode is to be recalled.
- **g_cic** = the CIC of the message, if the SI indicates a Service which contains the CIC in the MTP. Otherwise **g_cic** = **undefined_cic**

Entry/Exit Conditions for Service Part Decodes

On Entry:

- **g_recall** = false
- **g_start_display** = 0
- **g_user_data_present** = false
- **g_user_data_SSN** = **undefined_SSN**
- **g_message** contains the message to be decoded.
- **g_message_length** contains the length of the message in octets.
- **g_octet_number** is the index within **g_message** of the first octet of the portion of the message which the called procedure should decode.
- **g_service_indicator** contains the Service Indicator field extracted from the message.
- **g_subservice_field** contains the Sub-Service Field field extracted from the message.
- **g_cic** = the CIC of the message if the Service Indicator is that of a Service which contains the CIC in the MTP, otherwise **g_cic** = **undefined_cic**
- **g_octet_info.segment** contains a code which must be set into **g_octet_decode_info^[x].segment**, to allow the correct segment to be loaded when **g_octet_decode_info^[x].proc** is called.

On Exit:

- **g_octet_decode_info** should have been set up as far as **g_octet_number-1**.
- **g_octet_number** is the index within **g_message** of the next octet to be decoded.
- **g_start_display** contains the octet number at which the display will start, if the Start Display (SD) option is enabled in the Personality file. Setting this is optional, it can be left at 0 as on entry.
- If **g_user_data_present** = TRUE, then **g_user_data_SSN** may be set with the SSN extracted or deduced from the message. If no SSN can be deduced, then it should be left as **undefined_SSN**.
- If **g_user_data_present** = TRUE, then **g_user_data_length** must contain the length of the user data parameter.
- **g_recall** = TRUE if the decode is to be recalled.

Entry/Exit Conditions for Userdata Part Decodes

On Entry

- **g_recall** = false
- **g_start_display** retains the value it had on exit from the Sevice decode.
- **g_message** contains the message to be decoded.
- **g_message_length** contains the length of the message in octets.
- **g_octet_number** is the index within g_message of the first octet of the portion of the message which the called procedure should decode.
- **g_octet_info.segment** contains a code which must be set into **g_octet_decode_info^[x].segment**, to allow the correct segment to be loaded when **g_octet_decode_info^[x].proc** is called.

On Exit

- **g_octet_decode_info** should have been set up as far as **g_octet_number-1**.
- **g_octet_number** is the index within **g_message** of the next octet to be decoded.

- **g_start_display** contains the octet number at which the display will start, if the alternative Start Display (SD) option is enabled in the Personality file. Setting this is optional, it can be left as on entry.

Entry/Exit Conditions for Recalled Decodes

On Entry

- **g_message** contains the message to be decoded.
- **g_message_length** contains the length of the message in octets.
- **g_octet_number** is the index within **g_message** of the first octet of the portion of the message which the called procedure should decode.
- **g_octet_info.segment** contains a code which must be set into **g_octet_decode_info^[g_octet_number].segment** to allow the correct segment to be loaded when **g_octet_decode_info^[g_octet_number].proc** is called.

On Exit

- **g_octet_decode_info** should have been set up as far as **g_octet_number-1**.
- **g_octet_number** is the index within **g_message** of the next octet to be decoded.

Field Decodes

The Decode Display

If a field decode procedure which has not yet been written is called, the decode area of the screen is blank (columns 18 - 80). This is the area you will write, by calling library procedures in module DA_WRITE.

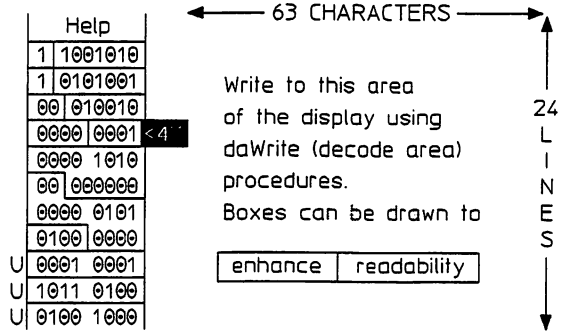


Figure 7-4.

Writing the Display

General Comments

Regard the decode area as a simple 24 line by 63 character display.The first line you write starts at column 18 on the top line of the screen.

All positioning is arranged by the DA_WRITE procedures you use.

Lines longer than 63 characters wrap-around and disturb the binary display on the left side of the screen.

For multi-octet fields, use the following technique to display more than 24 lines. Write two or more field decode procedures. Set name of the first one in **g_octet_info.proc** for the early octet(s) of the field, and the subsequent name(s) for the later octets. To view the first part, select one of the earlier octets and press D. To view the next part, select one of the later octets and press D.

Note

DO NOT ALTER ANY GLOBAL VARIABLES.

ALL SCREEN WRITES BY FIELD DECODE PROCEDURES MUST BE DONE VIA PROCEDURES FROM MODULE DA_WRITE.

DA_WRITE Procedures

These procedures control writing to the decode area of the screen, columns 18-80 inclusive. They are modelled on PASCAL IO.

Some procedures (for example **daWrite** and **WriteNum**) only cause data to be buffered ready for output, while others (for example **daWriteln** and **daWritelnStrNum**) cause any buffered data, plus their own parameters, to be output as a complete line.

At any time, **daNewline** can be called to output any buffered information, then take a new line.

The full list of procedures which only cause output to be buffered are:

daWrite;
daWriteUnderlined;
daWriteInverted;
daWriteNum;
daWriteIndented.

Combinations of these procedures can be called to build up a line from several small elements. **daNewline** can then be called to output the complete line.

Note

Ensure that the last call of a field decode is to **daNewLine**, or another procedure which causes output. If you do not, the last line of data will not be displayed.

PROCEDURE dlCannot Decode ;

This procedure is used with octets for which no field decode is available. It displays a screen with the text:

```
No further decode available for this octet
```

and gives the octet value in decimal and Hexadecimal.

Note

You can reduce the development time of the first version of a decode by using **dlCannotDecode** instead of a Field Decode Procedure for the simpler fields in your protocol. Later you can substitute field decodes for **dlCannotDecode** as required.

Simple Output Procedures

The **da_WRITE** module contains procedures for outputing simple items, such as numbers and character strings, and also many more complex features such as multi-line boxes with optional labels.

PROCEDURE **daTitle (char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and underlined. *This must be the only write to this line.*

PROCEDURE **daWrite (char_string : string80)** ;

PROCEDURE **daWriteln (char_string : string80)** ;
Writes **char_string** to the current position in the decode area, then moves on to the next line. Writes **char_string** to the current position in the decode area.

PROCEDURE **daNewline** ;
Moves on to the next line in the decode area. (Can be used to leave a blank line.) PROCEDURE **daWriteUnderlined (char_string : string80)** ;
Writes **char_string** to the current position in the decode area, and underlines it.

PROCEDURE **daWriteInverted (char_string : string80)** ;
Writes **char_string** to the current position in the decode area in inverse video.

PROCEDURE **daWriteIndented (indent : byte ; char_string : string80)** ;
Writes **char_string** to the current position in the decode area, indented by indent character positions from left of area.

PROCEDURE **daWriteNum (number : integer ; width : byte)** ;
Writes **number** to the current position in the decode area. **Number** is

written in a field **width** characters wide and right justified. If **number** is wider than **width**, it is written into a field of the minimum width possible.

PROCEDURE **daCentre (char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred. *This must be the only write to this line.*

PROCEDURE **daCentreUnderlined (char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and underlined. *This must be the only write to this line.*

PROCEDURE **daWritelnUnderlined (char_string : string80)** ;
Writes **char_string** to the current position in the decode area, underlines it, then moves on to the next line.

PROCEDURE **daWritelnInverted (char_string : string80)** ;
Writes **char_string** to the current position in the decode area, in inverse video, then moves on to the next line.

PROCEDURE **daWritelnIndented (indent : byte ; char_string : string80)** ;
Writes **char_string** to the current position in the decode area, indented by indent character positions from left of area, then moves on to the next line.

PROCEDURE **daWritelnStrNum (char_string : string80 ; number : integer ; width : byte)** ;
Writes **char_string, number** to the current position in the decode area, then moves on to the next line. **number** is written in a field **width**

characters wide and right justified. If **number** is wider than **width**, it is written in a field of the minimum width possible.

PROCEDURE **daWriteln2str(char_string1 : string80 ; char_string2 : string80)**;
Writes **char_string1, char_string2** to the current position in the decode area, then moves onto the next line.

PROCEDURE **daWriteln3 (char_string1 : string80 ; number : integer ; width : byte ; char_string2 : string80)** ;
Writes **char_string1, number, char_string2** to the current position in the decode area, then moves on to the next line. **number** is written in a field **width** characters wide and right justified. If **number** is wider than **width**, it is written in a field of the minimum width possible.

Outputting to Boxes

The following set of procedures handle the output of strings to boxes. The initial line of a box should be output using **daBox, daBoxSides, daBoxTop, daLabBox** or **daLabBoxSides.**

The size of the string sent to these procedures determines the width of the box. Subsequent calls to **daContinueBox, daContinueBoxSides, daContinueLabBox, daContinueLabBoxSides**, will centre their string parameter in a box of the established width and indentation.

Items written to boxes are underlined unless the procedure name has the word 'sides' in its name, in which case the line is not underlined. This allows multi-line items.

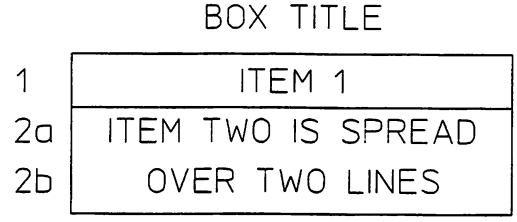
Items in boxes can be labeled using **da..LabBox** procedures. The box itself is centred, and the label appears immediately to the left of the box. Labeled and non-labeled lines can be mixed freely within the same box.

Complete sections of a box can be output in inverse video, by surrounding the calls which write them with calls to **daBoxInvertOn** and **daBoxInvertOff.**

Example of the use of some of the **daBox** ... procedures:

```
daBoxTop                ( '        BOX TITLE        ' ) ;
daContinueLabBox        ( '1  ', 'ITEM 1' ) ;
daContinueLabBoxSides ( '2a ', 'ITEM TWO IS SPREAD' ) ;
daContinueLabBox        ( '2b ', 'OVER TWO LINES' ) ;
```

will produce;



PROCEDURE **daBox (char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and in a box. This must be the only write to this line.

The procedure does not insert spaces inside the box, you must insert them.

PROCEDURE **daBoxTop (char_string : string80)** ;
Writes the title line for a following box.

Writes **char_string** to the current line in the decode area, centred and underlined. This must be the only write to this line.

PROCEDURE **daLabBox (lab_string : string80 ; char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and in a box. This must be the only write to this line.

lab_string is output right-justified preceding the box. The procedure does not insert spaces inside the box, you must insert them if required.

PROCEDURE **daBoxSides (char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and between box sides (but not underlined). This must be the only write to this line.

The procedure does not insert spaces inside the box, you must insert them.

PROCEDURE **daLabBoxSides (lab_string : string80 ; char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and between box sides (but not underlined).

lab_string is output right-justified preceding the box. The procedure does not insert spaces inside the box, you must insert them.

This must be the only write to this line.

PROCEDURE **daContinueBox (char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and in a box of the current width. This must be the only write to this line.

PROCEDURE **daContinueBoxPart (char_string : string80 ;udl_left_part : boolean ; udl_posn : byte)** ;
Writes **char_string** to the current line in the decode area, centred and in a box of the current width.

IF **udl_left_part** = TRUE, **char_string** is underlined from the start of the box to character **udl_posn** of **char_string**.

IF **udl_left_part** = FALSE, **char_string** is underlined from the character after **udl_posn** of **char_string** to the end of the box.

This must be the only write to this line.

PROCEDURE **daContinueLabBoxPart (lab_string : string80 ; char_string : string80 ; udl_left_part : boolean ; udl_posn : byte)** ;
Writes **char_string** to the current line in the decode area, centred and in a box of the current width.

lab_string is output right-justified preceding the box.

IF **udl_left_part** = TRUE, **char_string** is underlined from the start of the box to character udl_posn of char_string.

IF **udl_left_part** = FALSE, **char_string** is underlined from the character after **udl_posn** of **char_string** to the end of the box.

This must be the only write to this line.

PROCEDURE **daContinueLabBox (lab_string : string80 ; char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and in a box of the current width.

lab_string is output right-justified preceding the box.

This must be the only write to this line.

PROCEDURE **daContinueBoxSides (char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and in a box of the current width. This must be the only write to this line.

7

PROCEDURE **daContinueLabBoxSides (lab_string : string80 ; char_string : string80)** ;
Writes **char_string** to the current line in the decode area, centred and between box sides of the current width.

lab_string is output right-justified preceding the box.

This must be the only write to this line.

PROCEDURE **daBoxInvertOn**

This procedure causes all subsequent lines written to a box to be inverted, until the next call of **daBoxInvertOff**.

PROCEDURE **daBoxInvertOff**
This procedure turns off the invertion of strings written to boxes.

List of Global Variables Used by Decodes

g_message (READ ONLY). Contains the message to be decoded.

g_message_length (READ ONLY). The length of the message to be decoded.

g_octet_number The current position within **g_message**. Updated by EVALUATION procedure for each octet it decodes.

g_service_indicator, g_subservice_field Set by MTP decodes and used to select the Service decode.

g_user_data_present Set to TRUE by Service decode to indicate the presence of UserData which needs to be decoded. If a decode sets this TRUE, it must also set up **g_user_data_SSN** and **g_user_data_length.**

g_user_data_SSN Set by Service decodes if a message contains UserData, and used to select the UserData part decode to be used.

g_user_data_length Set by Service decodes if a message contains UserData, and used by the UserData part decode. This is needed because in some cases the length cannot be deduced from the contents of the UserData part.

g_recall Set by <decode_name>EVALUATION to cause **<decode_name>RECALL** to be called after the UserData has been decoded.

g_start_display contains the octet number at which the display will start, if the alternative Start Display (SD) option is enabled in the Personality file. Setting this is optional, it can be left as on entry.

g_H0 This allows the EVALUATION part of a decode to store the value of the H0 field for use by FIELD decode procedures.

g_H1 This allows the EVALUATION part of a decode to store the value of the H1 field for use by FIELD decode procedures.

g_message_type This allows the EVALUATION part of a decode to store the value of the MT field for use by FIELD decode procedures.

g_user_0 to **g_user_9** These enable the EVALUATION part of user-defined decodes to pass information to the corresponding FIELD decode procedures.

g_octet_info This is used to build decode information for each single octet.

g_octet_decode_info The information built in g_octet_info for each octet (g_octet_number) is then copied to g_octet_decode_info^[g_octet_number].

User-Defined Variables (g_user_n)

It might be necessary to set up more information than that contained in **g_octet_decode_info** for use by field decode procedures.

Ten variables have been reserved for use by user-defined decodes. They are all of type **integer**, and are named **g_user_0** , **g_user_1** .. **g_user_9** .

Library Procedures Supplied by Hewlett-Packard

The following library procedures use the TYPE definitions.

```
byte = 0..255;
str1 = string [1];
string32 = string [32]; string80 = string [80];
```

These types are included in the IMPORT library files, and do not need to be defined by the user-defined decode.

7

DLIBRY Procedures

The following procedures are also available for use in Field Decode Procedures.

FUNCTION dlHex(value : INTEGER): Str1;
The function **dlHex** calculates the character corresponding to **value**, via Hexadecimal coding. For values of **value** outside the range 0 .. 15 the character 7 is returned.

FUNCTION HexString(startpos : INTEGER; num_digits : INTEGER) : string80;

The function **HexString** returns a string corresponding to **num_digits** Hex digits, taken from the message being decoded. Each Hex digit is given by 4 bits, starting at **g_message[startpos]** (start of an octet).

FUNCTION dlIa5(value : INTEGER) : Str1;
The function **dlIa5** returns a character corresponding to **value**, via International Alphabet No.5 coding. The character set is that of the "Primary set of graphic characters", as in TABLE 1/T.51 in Red Book VII.3 "Terminal Equipment and protocols for Telematic Services".

Non-printable characters, and values outside the range of Ia5 coding are returned as null.

FUNCTION ExtractField(from_octet : INTEGER ; start_bit : byte ; width : byte) : byte ;
Function to return **width** bits extracted from **from_octet**, with **start_bit** being the leftmost bit to be extracted. The bits are numbered 8(MSB) .. 1(LSB) . The returned value is right-justified.

FUNCTION ForHex(NumberOfBits : INTEGER) : INTEGER ;
Return the number of Hexadecimal digits needed to display a number of **NumberOfBits** of Binary digits.

FUNCTION ForDecimal(NumberOfBits : INTEGER) : INTEGER ;
Return the number of Decimal digits needed to display a number of **numberOfBits** of Binary digits.

FUNCTION ConvertStr(Base ,
Value ,
NumberOfDigits : INTEGER ;
StripLeading0s : BOOLEAN) : string32 ;

Return a string of size **NumberOfDigits** containing the representation of **Value** in **Base**.

FUNCTION BinStr(Value , NumberOfDigits : INTEGER) : string32;
Return a string of size **NumberOfDigits** containing the Binary representation of **Value**. Do NOT strip leading zeroes.

FUNCTION HexStrNo0s(Value , NumberOfDigits : INTEGER) : string32;
Return a string of size **NumberOfDigits** containing the Hexadecimal representation of **Value** with leading zeroes stripped.

FUNCTION HexStr(Value , NumberOfDigits : INTEGER) : string32;
Return a string of size **NumberOfDigits** containing the Hexadecimal representation of **Value** with leading zeroes retained.

FUNCTION DecStr(Value , NumberOfDigits : INTEGER) : string32;
Return a string of size **NumberOfDigits** containing the Decimal representation of **Value** with leading zeroes stripped.

Index

Hewlett-Packard Sales and Service Offices

US FIELD OPERATIONS HEADQUARTERS
Hewlett-Packard Company
19320 Pruneridge Avenue
Cupertino, CA 95014, USA
(800) 752-0900

California
Hewlett-Packard Co.
1421 South Manhattan Ave.
Fullerton, CA 92631
(714) 999-6700

Hewlett-Packard Co.
301 E. Evelyn
Mountain View, CA 94041
(415) 694-2000

Colorado
Hewlett-Packard Co.
24 Inverness Place, East
Englewood, CO 80112
(303) 649-5000

Georgia
Hewlett-Packard Co.
2000 South Park Place
Atlanta, GA 30339
(404) 955-1500

Illinois
Hewlett-Packard Co.
5201 Tollveiw Drive
Rolling Meadows, IL 60008
(708) 255-9800

New Jersey
Hewlett-Packard Co.
120 W. Century Road
Paramus, NJ 07653
(201) 599-5000

Texas
Hewlett-Packard Co.
930 E. Campbell Rd.
Richardson, TX 75081
(214) 231-6101

EUROPEAN OPERATIONS HEADQUARTERS
Hewlett-Packard S.A.
150, Route du Nant-d'Avril
1217 Meyrin 2\Geneva
Switzerland
(41 22) 780.8111

France
Hewlett-Packard France
1 Avenue Du Canada
Zone D'Activite De Courtaboeuf
F-91947 Les Ulis Cedex
France
(33 1) 69 82 60 60

Germany
Hewlett-Packard GmbH
Berner Strasse 117
60000 Frankfurt 56
West Germany
(49 69) 500006-0

Great Britain
Hewlett-Packard Ltd.
Eskdale Road,
Winnersh Triangle
Wokingham,
Berkshire RG11 5DZ
England
(44 734) 696622

INTERCON OPERATIONS HEADQUARTERS
Hewlett-Packard Company
3495 Deer Creek Rd.
Palo Alto, California 94304-1316
(415) 857-5027

Australia
Hewlett-Packard Australia Ltd.
31-41 Joseph Street
Blackburn, Victoria 3130
(61 3) 895-2895

Canada
Hewlett-Packard (Canada) Ltd.
17500 South Service Road
Trans-Canada Highway
Kirkland, Quebec H9J 2X8
Canada
(514) 697-4232

Japan
Yokogawa-Hewlett-Packard Ltd.
1-27-15 Yabe, Sagamihara
Kanagawa 229, Japan
(81 427) 59-1311

China
China Hewlett-Packard, Co.
38 Bei San Huan X1 Road
Shuang Yu Shu
Hai Dian District
Beijing, China
(86 1) 256-6888

Singapore
Hewlett-Packard Singapore
Pte. Ltd.
1150 Depot Road
Singapore 0410
(65) 273 7388

Taiwan
Hewlett-Packard Taiwan
8th Floor, H-P Building
337 Fu Hsing North Road
Taipei, Taiwan
(886 2) 712-0404

WIN AN HP CALCULATOR!

Fill in the Reader Comment Card supplied opposite; return it to HP and you could win an HP Calculator.

Every three months, all returned cards will be entered into a free prize draw for either an HP Scientific or HP Business Calculator.

Note

When you fill in the card, remember to tick the box showing which type of calculator you would prefer if you win.

Hewlett-Packard reserves the right to withdraw this offer at any time.

Reorder No.
37900-90090

Manufacturing Part No.
37900-90090