Job Control and IOF

Bull DPS 7000

Programmer's Manual

IOF Programmer's Manual

GCOS 7

Software

Subject : This manual describes the features of GCOS 7 which a programmer developing programs for interactive execution needs to know.

Special instructions : This manual replaces the document referenced 47 A2 05UJ Rev05 for all users of GCOS 7 Release V7. 47 A2 05UF remains valid for users of GCOS 7 V6.

Software supported : GCOS 7-AP/HPS/EXMS Release V7

Date : February 1995

Bull S.A.
CEDOC
Atelier de reproduction
331, Avenue Patton BP 428
49005 ANGERS Cedex
FRANCE

Bull HN Information Systems Inc.
Publication Order Entry
MA30/843
300 Concord Road
Billerica, MA 01821
U.S.A.

47 A2 37UJ Rev00

Preface

MANUAL OBJECTIVES

This manual describes those features of GCOS 7 with which a programmer developing programs for interactive execution must be familiar.

INTENDED READERS

The intended readers of this manual are programmers who wish to create and/or maintain programs that execute interactively under IOF (Interactive Operation Facility). A general knowledge of GCOS 7 (such as may be obtained by reading Part 1 of the *IOF Terminal User's Reference Manual*), is assumed, as well as familiarity with COBOL, FORTRAN, C, or GPL.

STRUCTURE OF THIS MANUAL

Section 1	introduces the concepts associated with executing programs interactively under IOF.
Section 2	describes Line mode and Forms mode, the two modes in which a terminal can be accessed.
Section 3	describes character sets, the values of which are used to code characters transmitted from or to a terminal.
Section 4	discusses the TAM (Terminal Access Method).
Section 5	discusses the SDPI (Standard Device Programmatic Interface).
Section 6	describes a few miscellaneous service primitives.

NOTE: Earlier versions of this manual included a section on the FORMS facility. This information is now contained in the *GCOS 7 Forms User's Guide*.

Earlier versions of this manual also included sections on GCL translation, services provided by GCL to interactive programs, GCL command management, and the creation and modification of Help texts. This information is now contained in the *GCL Programmer's Manual*.

ASSOCIATED DOCUMENTS

The following manuals are companion documents:

GCOS 7 Forms User's Guide	*47 A2 15UJ*
GCL Programmer's Manual	*47 A2 36UJ*

The following manual is prerequisite reading to understand the material presented in this manual:

IOF Terminal User's Reference Manual (Part 1)	*47 A2 38UJ*

The following manuals are also recommended:

IOF Terminal User's Reference Manual (Part 2)	*47 A2 39UJ*
IOF Terminal User's Reference Manual (Part 3)	*47 A2 40UJ*
GCOS 7 System Administrator's Manual	*47 A2 41US*
GCOS 7 Forms User's Guide	*47 A2 15UJ*

The following manuals should be consulted as needed.

COBOL 85 Reference Manual	*47 A2 05UL*
COBOL 85 User's Guide	*47 A2 06UL*
FORTRAN 77 Reference Manual	*47 A2 15UL*
FORTRAN 77 User Guide	*47 A2 16UL*
GPL Reference Manual	*47 A2 35UL*
GPL User Guide	*47 A2 36UL*
GPL System Primitives Reference Manual	*47 A2 34UL*
C Language User's Guide	*47 A2 60UL*
C Language System Primitives Reference Manual	*47 A2 64UL*

NOTATION CONVENTIONS

The following notation is used in the formats of the commands presented in this manual:

ITEM	Uppercase characters are used for keyword items that are to be entered exactly as shown.
item	Lowercase characters are used for user-supplied items.
[item]	An item within square brackets is optional.
{item1} {item2} {item3}	A column of items within braces means one item must be selected. The default, if there is one, is underlined (e.g., item3 here).
{item1|item2|item3}	A sequence of items separated by vertical bars has the same meaning as the convention just above.
...	An ellipsis indicates that the preceding item may be repeated one or more times.

Table of Contents

Illustrations

1. Introduction

1.1 IOF AND BATCH COMPATIBILITY

The Interactive Operating Facility (IOF) available under GCOS 7 offers many powerful programming features, but remains largely compatible with traditional Batch programming. Among other things:

- the programming tools (compilers, linker, editors, debugger, etc.) are the same for IOF and Batch,
- programs are interchangeable: programs written with IOF can be run in Batch and vice versa (except when dialog is with a terminal, as in FORMS mode for example)
- with IOF, as in Batch, a programmer has access to all files without restriction through the various access methods,
- in both IOF and Batch programming, advanced GPL primitives are available,
- all GCOS 7 resources are equally available to IOF and to Batch, including:
    - cataloged, uncataloged, and temporary files,
    - variable memory (SIZE parameter of the EXEC_PG command),
    - backing store space,
    - CPU.

1.2 INTERACTIVE OPERATOR CONTEXT

The interactive facilities available under GCOS 7 are intended to:

- provide a user-friendly environment,
- maximize programmer productivity,
- provide for easy programming,
- be flexible enough to cater for different (and possibly conflicting) end-user requirements.

Part 1 of the *IOF Terminal User's Reference Manual* is a prerequisite to this manual, because it gives an overview of IOF facilities.

Under IOF the terminal user is provided with menus, prompts, and help texts, to explain each domain, command, and parameter. You can dialog with the system in full screen mode when using a terminal which supports forms. You are provided with a specific profile which allows you to define your own working mode and resource requirements.

The programmer creating interactive applications to run under IOF is expected to provide end-users (terminal operators) of the applications with the same level of facilities.

1.3 INTERACTIVE PROGRAM CONTEXT

All the GCOS 7 tools useful to the programmer can be activated interactively. These tools include library maintenance, editors (both the line editor and the full-screen editor), compilers, the linker, and the debugger.

These facilities are described in the *IOF Terminal User's Reference Manual* and in the manual(s) specific to each product.

Working environments may be made available by the System Administrator. Standard environments for IOF and batch programming are provided with GCOS 7, as described in the *GCOS 7 System Administrator's Manual*.

The following commercial programming languages are available: COBOL, FORTRAN, PASCAL, C, and GPL. The compilers for these languages are available both under IOF and in batch.

IQS is discussed in its own manual set and is not treated here.

1.4 INTERACTIVE PROGRAM SPECIFICS

The new features specific to programming under IOF are designed to facilitate the interaction between a program and its user(s) during the execution of the program.

The main difference between an interactive program and a batch program is that the former interacts with a user (terminal operator) during its execution. Thus the specific needs of an IOF application programmer are for facilities which handle a dialog with a terminal at different levels and in various modes.

The object of this manual is to present these facilities.

1.5 TERMINALS

The modes in which a program can dialog with a terminal operator depend on the characteristics of the terminal.

A list of the terminals supported by GCOS 7 is presented in Table 1-1 below. Terminals which support Forms Mode (that is, permit a dialog in full-screen mode) are marked with an "F" in the table. Terminals which support the PLW character set (described in Section 3) are indicated with a "P" in the table.

TTY		VIP synchronous		BSC 3270	
DKU7001/2		DKU7005	F	IBM3270	F
MINITEL	FP	DKU7007	F	IBM3741	
PC7800	F	DKU7105	FP	IBM3278	F
PRT1220	P	DKU7107	FP	IBM3279	F
TTU8124		DKU7211	FP		
TTU8126		PDP11/10			
TTU8128		TTU8221			
TTY33		VIP7804	F		
TTY35		VIP8800	FP		
TWS2255	FP				
VIP7801	F				

Figure 1-1. Terminals supported by GCOS 7

1.6 USING THE TERMINAL

As explained above, the specific features which are of interest to the IOF applications programmer concern dialog with the terminal.

A large number of primitives and commands are available to assist in programming under IOF. These are described in detail later in this manual; there is a presentation of each primitive in terms of its function, format, parameters, return codes, restrictions (if any), and examples of use.

Various levels of communication with the terminal are available, as discussed below.

1.6.1 Basic Level

The basic level enables the program to exchange information with the terminal for most general purposes. At this level, two modes are provided: line (or text) mode, and forms mode.

1.6.1.1 LINE MODE

In line mode, text is input or output in continuous streams; this mode is designed for source text processing. You might use this mode for entering a source program written in COBOL, PASCAL, C, GPL, or FORTRAN, or for entering text such as the contents of this manual. In this mode, the interface between the terminal and the rest of the system and processors is the Terminal Access Method (TAM), which gives access to the terminal as if it were a conventional file.

The TAM interface is discussed in Section 4 of this manual.

1.6.1.2 FORMS MODE

In forms mode, the terminal screen is composed of one or more fixed parts - the forms - that contain variable fields to be filled in by the user or user program. This mode can be used for commercial formats and is particularly good for graphically presenting information and prompting a user response. The interface between the terminal and the rest of the system and processors is the Standard Device Programmatic Interface (SDPI).

The SDPI interface enables you to activate forms from within a program. These forms are used in the dialog with the terminal. You can create (and store) such forms using the MAINTAIN_FORM processor.

The SDPI interface is discussed in Section 5 of this manual.

1.6.1.3 MAINTAIN_FORM PROCESSOR

You can design forms directly on a terminal (using the ICREATE command within the processor), or you can specify the form using a COBOL-like source form definition language. Both these ways of creating forms are described in the *GCOS 7 Forms User's Guide*.

This processor takes its input, either directly from a terminal, or from a library member (or file). However, you are recommended to create forms directly using the ICREATE command of the MAINTAIN_FORM processor.

You can use commands such as MODIFY, TEST, and EDIT or FSE, to test, debug, and modify forms. For details, see the *GCOS 7 Forms User's Guide*.

1.6.2 GCL Interface

The GCL interface enables you to use the GCL (GCOS Command Language) to handle the dialog with the terminal. This command language is described in detail in the *IOF Terminal User's Reference Manual*.

GCL is recommended for all applications that accept large amounts of complex information from the terminal.

The GCL interface provides facilities such as command prompting, Help text invocation, and error recovery mechanisms without any special programming effort.

The GCL interface is available in both line and forms mode. Thus, no specific restrictions apply to the characteristics of the user's terminal for the use of this interface. This interface is discussed in the *GCL Programmer's Manual*.

1.6.2.1 MAINTAIN_COMMAND PROCESSOR

The command management processor (MAINTAIN_COMMAND) handles the creation, modification, and compilation of GCL commands written at your installation. This processor is discussed in the *GCL Programmer's Manual*.

The creation and maintenance of Help texts associated with GCL commands and their parameters is discussed in the *GCL Programmer's Manual*.

1.6.2.2 SERVICES

The GCL interface provides various services which the IOF programmer can use to facilitate his task. These include the following:

- calling up Help texts from within an IOF program,
- reading and/or setting system and global variables to test and/or modify the operating environment,
- analyzing file and fileset literals input from the terminal.

You can define your own GCL commands, and have them compiled to be used in your programs (see the *GCL Programmer's Manual*).

You can tailor a procedure to suit your own personal needs exactly. You can also compress a series of existing procedures and their parameter values into a single procedure. A simple, step-by-step example of using the menus of MAINTAIN_FORM to create and modify a form, is described in the *GCOS 7 Forms User's Guide*.

A few miscellaneous service primitives are presented in Section 6 of this manual.

2. Line Mode and Forms Mode

2.1 OVERVIEW

Under IOF, there are two modes in which a terminal can be accessed. These two modes are Line mode (also called Text mode) and Forms mode.

Line mode is where data is input or output as a continuous stream of discrete units, either in blocks or character by character. It is the simpler mode, but is well adapted to editing source programs or text processing. Line mode is available on both TTY and block buffered terminals. In the case of Line mode, the interface handling the liaison between the user at the terminal and the program is the Terminal Access Method (TAM). With this access method, the terminal is seen as at least two separate files (an input file and an output file). Information is moved to and from these files.

Forms mode is where the screen is considered as a form or a set of forms. Each form is composed of fixed and variable fields. Only the variable fields are filled in by the user or program. Form mode is available only on some block buffered terminals. The interface used to handle this mode is the Standard Device Programmatic Interface (SDPI). Essentially, the terminal is seen as an object with which the program is conversing - sending and receiving information alternately.

The two access methods will be presented in Sections 4 and 5 of this manual, after further details on Line and Forms modes, below.

2.2 LINE MODE

TAM is the interface handling the link between the terminal in Line mode and the program. These aspects should be noted:

- TAM is fully compatible with other data management interfaces. Hence, data may be directed to or from the terminal (or a file), simply by assigning an internal file name to the terminal as you would do for a conventional file.
- Both SARF and SSF file formats are available.
- As regards output, the terminal is seen as the equivalent of a printer, so that for example, some of the features of the SYSOUT access method are available to the terminal.
- By using different internal file names, a logical separation can be made between the input and output of a program.

TAM is accessible through COBOL, FORTRAN, PASCAL, C, and GPL.

2.3 FORMS MODE

SDPI is the interface handling the terminal in Forms mode. The terminal itself must have the necessary features for the depiction of forms, for example, reverse video or high/low intensity differentiation, or the more advanced features of color. The *GCOS 7 Forms User's Guide* contains a full list of terminals which can support Forms mode.

SDPI is accessible through COBOL, C, and GPL. The SDPI interface is common to both TDS and IOF environments.

2.4 SHARING THE TERMINAL BETWEEN TAM AND SDPI

A terminal may be shared between TAM and SDPI. From the program's point of view, each interface has a different visibility, each managing an independent virtual terminal. At any one moment, there is only one virtual terminal mapped onto the actual terminal.

The system also keeps an image of the contents and attributes of the variable fields of a Form used on the terminal, and this information can be restored when passing back into Forms mode. So, for example, if a programmer in forms mode passes into Line mode and then back into Forms mode, the previous information (relative to Forms mode) has been stored by the system and is restored. From the point of view of the terminal, when a switching between logical terminals occurs (while a form is being displayed), these events take place:

- a synchronization message is issued, if necessary (the characters +++ are displayed at the bottom of the screen),
- the image on the screen is saved,
- the screen is cleared,
- the statement is executed.

When an SDPI primitive is subsequently issued;

- the stored form is displayed on the screen. Variable fields are restored to their previous values and attributes,
- the statement is executed.

3. Character Sets

3.1 EXTENDED CHARACTER SETS

The values used to code characters input from or output to your terminal belong to one of the following sets:

- C101 The standard EBCDIC character set.
- C114 The Arabic character set.
- C118 The Greek character set.
- C113 The Cyrillic character set.
- C094 The Chinese character set.
- PLW The PLW (Pluri Lingual West) character set (Figure 3-1).
- PLE The PLE (Pluri Lingual East) character set.

PLW (Pluri Lingual West) is a character set which contains most of the characters used in those Western European languages which use the Latin alphabet. The set includes the usual alphanumeric characters plus the accented characters. This character set is presented in Figure 3-1.

Note that:

- Hatched codes are actually part of the PLW character set but are not supported by DKU terminals or by VIP8800 family terminals.
- This figure does not show the terminal management characters, for instance FW (code 12), BS (code 16), SYN (code 32). These characters do not belong to the PLW character set.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					▽	&	-	⌀	∅	°	µ	¢	{	}	\	0
1						é	/	É	a	j	~	£	A	J	÷	1
2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
3					ä	ë	Ä	Ë	c	l	t	'	C	L	T	3
4					à	è	À	È	d	m	u	○	D	M	U	4
5					á	í	Á	Í	e	n	v	§	E	N	V	5
6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
9					ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
A					[	]	¦	:	«	ª	¡	⌉		1	2	3
B					.	$	,	#	»	º	¿	|	ô	û	Ô	Û
C					<	*	%	@	ò	œ	o	‾	ö	ü	Ö	Ü
D					(	)	_	'	ý	,	ý	¨	ò	ù	Ò	Ù
E					+	;	>	=	þ	Æ	þ	'	ó	ú	Ó	Ú
F					!	^	?	"	±	¤	○	x	õ	ÿ	Õ	

Figure 3-1. PLW Characters (EBCDIC)

The way that TAM and SDPI handle PLW characters that do not belong to the C101 common subset depends on these two factors:

- whether or not the terminal supports PLW. (These terminals support PLW: MINITEL, DKU7105, DKU7107, DKU7211, PRT1220, TWS2255, and VIP8800.)
- the value of the #CSET system variable. The default is 1, for PLW, but see below. This variable is an element of the user profile. Its value may be set either from within the program (using MODVAR, which is described in the *GCL Programmer's Manual*) or by the terminal operator using the MODIFY_PROFILE directive.

The normal combinations are the following:

- #CSET=0 (C101) for an application program which deals only with the C101 character set. In this case, presentation control will prevent the terminal operator from entering invalid characters even if his terminal supports PLW.
- #CSET=1 (PLW) with an application program that uses the extended character set even if the user's terminal does not support PLW.

The table shows how PLW characters that do not belong to the C101 common subset are dealt with under various conditions.

Input or Output Mode	C101 mode (#CSET=0)		PLW mode (#CSET=1) or PLE mode (#CSET=7)	
	VIP8800	other terminals	terminals do not support PLW (or PLE)	terminals support PLW (or PLE)
input	rejected	rejected	not possible (character not on keyboard)	translated from interchange coding to direct EBCDIC
output	replaced by the "invalid character"		folded, i.e., translated from PLW EBCDIC	translated from direct EBCDIC to interchange coding

In addition, the special character sets can be selected by using the following values for #CSET:

- #CSET = 2 (APL).
- #CSET = 3 (C114) Arabic character set.
- #CSET = 4 (C118) Greek character set.
- #CSET = 5 (C113) Cyrillic character set.
- #CSET = 6 (C094) Chinese character set.
- #CSET = 7 (PLE) Pluri Lingual East character set.

PLE (Pluri Lingual East) is a character set which contains most of the characters used in those Eastern European languages which use the Latin alphabet. The set includes the usual alphanumeric characters plus the accented characters, letters with cedilla, diacritical marks, the stroke, etc. PLE is, like PLW, an extended character set and subject to the same considerations.

3.1.1 C101 Mode

The terminal is assumed to operate in 95-character (C101) mode. Where possible, presentation control forces it to operate in this mode.

A program which operates in C101 mode can send/receive only characters which belong to the C101 subset of EBCDIC. All other characters are considered to be invalid characters and they are replaced by a substitution character. This substitution character is contained in the system variable #INVCHAR. The value of this variable can be modified either by the terminal operator, or by the program. The default value of this variable is ".".

3.1.2 PLW Mode

Extended character set usage is the same for PLW and PLE. In the following discussion of PLW, it is assumed that the same considerations hold true for PLE.

When a program operates in PLW mode, it is allowed to send and receive all the characters of the PLW direct code. If the terminal concerned supports PLW, then it is assumed to be operating in this mode and the characters are displayed in PLW code.

If the terminal concerned does not support PLW, then any PLW characters that do not belong to the C101 common subset are "folded". In folding, each accented letter is replaced by its equivalent non-accented letter. The purpose of folding is to avoid sending invalid characters (which is the alternative to folding). However, folding does not guarantee that the resulting text is intelligible in all cases.

When a form of type PLW is activated by means of a CDGET primitive and the program concerned operates in PLW mode, PLW characters in the object form are displayed in PLW code (if the terminal supports PLW) or are folded into C101 characters (if the terminal does not support PLW).

When a form of PLW type is moved to a diskette by MAINTAIN_FORM, folding is performed systematically.

4. Terminal Access Method

4.1 HOW IT WORKS

TAM is an access method designed to support an interactive program, and yet remain consistent with classical GCOS access methods. TAM is designed as a data management interface to handle the exchanges between an interactive program and a terminal.

This compatibility of interactive and file contexts has these features:

- a program accessing a sequential file through the COBOL verbs OPEN/READ/WRITE/CLOSE is executable at an IOF terminal without further modification,

- conversely, a program written for an interactive context can use the file verbs OPEN/READ/WRITE/CLOSE,

In addition, the GPL H_TN primitive permits the user to dynamically redefine several options of the TAM. (Note however, that in a file context, the H_TN primitive is refused with the FUNCNAV return code ; i.e., function not available.)

Further details on these primitives are given below.

- an internal file name, ifn, is assigned to an IOF terminal by specifying TN as the device class in the ASGi (file assignment) parameter group of the EXEC_PG command.

The user may assign several ifns to the same terminal. A program needs at least two ifns for interactive dialog - one for input and one for output.

The ifn is the means by which a program specifies terminal handling and presentation characteristics, which are independent from those of other ifns. These characteristics include data format, default prompts, etc.

There is, however, the problem that in using several ifns, various outputs will arrive on the screen in chronological order, with all ifns mixed, along with any other information sent to the terminal from other interfaces.

4.2 TWO IMPORTANT ASPECTS OF THIS INTERFACE

4.2.1 Quarantining

TAM can block successive WRITEs to ifns assigned to the same terminal in order to minimize the number of input/output operations. A user program can, nevertheless, override this control by forcing a WRITE to the terminal with the FORCEWRITE option.

- In general, TAM will trigger an information transfer in these cases:
- FORCEWRITE (an option of the H_TN primitive),
- WRITE (PUT) new page,
- READ (GET),
- switching to another mode (e.g., to Forms mode,)
- time out,
- screen full,
- buffer overflow.

4.2.2 Prefixing and Prompting

In addition to quarantining, prefixes and prompts can be used to associate particular ifns with information generated chronologically on the screen.

Output: Messages output on the screen may be prefixed. This prefix is a string, chosen by the programmer, of between 1 and 12 characters. By default, the prefix is 3 spaces.

The programmer can also have data output without a prefix.

Input: Here, the situation is slightly different. A prompt, unique to a particular ifn, can appear on the screen when input for that ifn is required.

Three prompt possibilities are provided:

- normal prompt: the prompt is limited to 12 characters specified either statically (H_FD) or dynamically (H_TN). The default prompt is the ifn concatenated with a colon (:) and a space,
- temporary prompt: the prompt is limited to the record size, and must be specified through an H_TN primitive. Its effect is on the next H_GET operation,
- no prompt: where the terminal is just as in input mode.

4.3 OUTPUT

An output file can be handled in one of two presentation modes: edited mode, or transparent mode. Each of these is described below.

The format of a file, either SSF or SARF, determines the output presentation of the records of this file. SARF is the simpler format, having records or lines one after the other. Each SSF record contains an 8-byte header and control records which give presentation information - page length, width, etc.

The essential idea is that in edited mode, presentation controls are not included in the data records of a file, and it is the IOF access method that handles the presentation, following given system and user variables. If particular sequences (e.g., escape sequences) must be directed to the terminal without any alteration, then the transparent mode is used. In this case, no check on the data is done by the access method.

The advantage of edited mode is that a programmer can code his program without having to pay attention to the terminal type. For example, a programmer can indicate that he wants the next record after a blank line, without having to prefix this record with "carriage-return-line-feed" (which is not effective on all terminals; e.g., IBM 3270).

4.3.1 Edited Mode

Screen Presentation

The program can specify the screen presentation either implicitly with SARF, where each record starts at the beginning of a new line, or in the SSF record headers and control records.

The data part of the record is not to contain presentation characters (carriage return, line feed, form feed). Any such character is translated into the current substitution character. This character is defined by the system variable #INVCHAR (the default is .).

IOF counts the number of lines sent from the program to the terminal and divides this output into a series of screens or pages. The length of each screen is determined in the user profile (MP command) by the system variables #PL (Page Length) and #PW (Page Width).

A +++ prompt appears on the bottom of the screen and waits for the user to press the TRANSMIT key to display the next screen.

Organization of output into screens is known as PAGEMODE and is standard for non-scrolling terminals. Scrolling terminals, however, may use this mode by assigning a system variable (#PAGEMODE=1).

The four system variables concerned with output, #INVCHAR, #PL, #PW, and #PAGEMODE are described in the *GCL Programmer's Manual*. Assigning these variables in the user's profile is discussed in the same manual.

Printer-like Presentation

By using the appropriate control records, SSF edited mode can be used to present data on the screen as on a line printer; i.e., with headings, footings, line numbering, etc. A programmer can specify these characteristics:

- the system variable #PL (Page Length) specifies the form height
- the system variable #INVCHAR specifies the alternative presentation of characters that cannot be displayed on the screen. The default is "."
- HOF (Head Of Form) specifies the line number of the first line on the screen to be used. The default is HOF=1
- FF1 (first level of Full Form) specifies the last line on the screen to be used. FF2 (second level of Full Form) specifies the line at which a return code message will be produced. These restriction applies: 1 <= HOF <= FF2 <= FF1. The defaults are FF1=FF2=#PL.

Line Folding

Depending on the format, TAM does the following:

- in SARF format, the line folding is automatic,
- in SSF format, truncation can be requested by a control record, but folding is generally specified through the system variable #PW. Folding is performed by splitting the record after the page width specified by #PW. This width must not exceed the physical printing width of the terminal.

Note that if the programmer wants to control the folding of an output text at a given point, he must generate two separate records. The data part of the records is supposed not to contain any backspace character as far as folding is concerned.

Tabulation

Tabulation stops can be set at the terminal, in order to speed up the display of output, and will be inserted automatically by TAM just before the physical output to the terminal.

Usually the output data does not contain any embedded tabulation characters. However, a Control Record (CR 131) can be used to specify the values of tabulation stops for expansion, as well as coding the tabulation character, for a particular ifn. Later on, IOF will perform the expansion of the subsequent data records according to the CR131 of that ifn. Note that the record length after expansion should not exceed the RECSIZE value. The expansion of tabulations takes place before line folding processing.

Files and Output Presentation

Obviously the presentation will be correct if only one ifn is used and if asynchronous messages are held during the display of output.

Therefore two typical uses of TAM are:

- one ifn in OUTPUT and another in INPUT are used to establish a dialog. The ifn for INPUT is used without any sophisticated presentation requirements,

- one ifn active for OUTPUT is used to provide a report with a neat presentation; i.e., headings, footings, margin, etc.

Any other situation with several ifns in OUTPUT at the same time, or one in INPUT and another in OUTPUT and requiring a neat presentation in OUTPUT, will give a rather unsatisfactory result due to the trade-offs that have to be made to support those contradictory requirements (such as heading/footing intermixing, or an automatic new page jump after an input).

4.3.2 Transparent Mode

Any binary sequence may be submitted by the program by specifying through a H_TN primitive that the next primitive will be in transparent mode.

Any control record is then ignored, so there is no processing of tabulations, line folding, etc. As regards counting of lines internally to maintain the cursor position, IOF assumes that the cursor position must be incremented by one line.

4.4 INPUT

4.4.1 Edited Mode

The data records which are delivered by the access method do not contain any presentation characters, such as carriage return, line feed, etc.

Therefore, there are several points the programmer should know as regards presentation in edited mode:

- Tabulation: by using the system variable, #EXPTABS, a programmer can choose between:
  - either having tabs automatically expanded (as blank characters) as text is entered. This is the default value for #EXPTABS,
  - or not having tabs expanded. They are passed embedded in the input text. Further, if the ifn is in SSF format, a special Control Record (CR 131) is generated.
- Text over several lines: the user can define a text continuation character by means of the system variable #CC, then use this character as the last character of an input line before transmitting the line. All lines entered this way are considered as parts of the same record. The programmer can eliminate this continuation facility by setting the #CC variable to a zero-length string.
- Files in SSF format: record headers and control records can be used in inputting files in SSF format - for example, including CR in the file when data is to be tabulated. The program must test the control/data record flag and check for a data record each time it receives a control record.
- Input data on a page mode basis: this facility is available only when inputting data in page mode on a synchronous terminal with RETURN and TRANSMIT keys. It means that several lines can be entered in a single transmission, but delivered record by record to the program. This avoids unnecessary swapping in and out, and prompting between two consecutive lines. All text entered as consecutive lines is saved, even if output messages must be displayed before this text has been completely processed by the program. However, a break command will cancel it being processed.

For further information on the system variables #CC, #EXPTABS and #TABS, refer to the *GCL Programmer's Manual.*

4.4.2 Transparent Mode

Each character entered at the terminal is transmitted to the program.

4.5 GCL TERMINAL ASSIGNMENT

The TAM allows an interactive program to access the terminal in a way fully compatible with sequential file access interfaces. Therefore the same program can run either with files or a terminal, depending on the environment and the GCL parameters used.

Note that several "files" of the same interactive program can share the same terminal, but there is no sharing a terminal between several programs.

An ifn is assigned to a terminal by using these parameters of the EXEC_PG GCL command:

- FILEi and ASGi parameters make this assignment.
- DEFi parameter can extend the file description with define parameters.
- ALCi parameter is not meaningful for the terminal, since no allocation is made.
- OUTi parameter is not meaningful since it only refers to a file to be written by the SYSOUT mechanism.

For a full description of the file parameter groups, refer to the *IOF Terminal User's Reference Manual*.

These data management options are the only options relevant to the TAM.

4.5.1 ASGi: File Assignment Parameters

```
ASGi = file-description
```

The most general file description is presented in the *IOF Terminal User's Manual*.

For the terminal, the file description is reduced to:

```
[full-path-name]: [md]: TN [$UNCAT]
```

- TN: the device class (terminal) refers to the main device of the terminal (keyboard/display or keyboard/printer),

- md: the media parameter, used to specify the name of the media,

- the external file name is not relevant to the main device of the terminal. The default value for the efn parameter is the ifn value.

4.5.2 DEFi: File-Define Parameter Group

This describes the file characteristics and the features particular to the TAM.

Significant parameters:

```
[ RECSIZE = dec5 ]

[ DATAFORM = { SARF | SSF } ]
```

Terminal file specific parameters:

```
[ PROMPT = char12 ]

[ { MSG | MSSG } = bool ]

[ EOF = char4 ]
```

4.6 GPL PRIMITIVES

All TAM primitives are presented schematically and in detail below.

The File Management primitives, which are not described here in this section, also apply to the TAM (e.g. H_CRFD, H_DLFD, H_ASSIGN, H_DEASSIGN).

NOTE: If a break occurs while a TAM file is open, a BREAK Return Code is returned with a subsequent primitive. Apart from the case of break, the return codes are the normal access method return codes.

4.6.1 H_FD (File Declaration)

Purpose:

H_FD is used to declare characteristics of a file. These can be updated by the ASGi or DEFi parameters of the EXEC_PG GCL command, or dynamically when the file is open, with the H_TN primitive.

Syntax:

```
$H_FD

      ifn

      [ { ACTUAL | EMPTY } [ ATTRIB = l_char ] ]

               { PROMPT = l_char12 }
      [ TN = ( {                   }  [ NMSG ] [ EOF=l_char4 ] ) ]
               { NPROMPT           }

      [ RECSIZE = l_digit5 ]

      [ BLKSIZE = l_digit5 ]

      [ DATAFORM = { SARF | SSF } ]

      [ REFMODE = { MOVE | LOCATE } ] ;
```

Parameters:

ifn	The internal file name, i_char8, is the name of a terminal file in a program or in GCL. This name must be unique within a compile unit, and within a linked group of compile units.
ACTUAL	Allocation has to be made for this file declaration, with initialization taking the user-provided parameters or system defaults.
EMPTY	No allocation is to be made for this file declaration. All existing parameters are ignored. (An ACTUAL declaration of the same ifn must be in another compile unit linked to this one).
ATTRIB	Attributes of a file are normally determined automatically, so the ATTRIB parameter can be omitted. There are, however, special cases where the BASED attribute is used with the EMPTY parameter. See the *GPL System Primitives Reference Manual*.
TN	Specifies the TAM parameters:
PROMPT	A string of 1 to 12 characters: - In input this string is sent to the terminal to indicate that an entry can be made (i.e., a H_GET primitive has been issued). The default prompt is the ifn string concatenated with a colon and a space; that is, "ifn: ". - In output, this string is used as an identifier prefix to the text displayed. The default prompt is 3 spaces.
NPROMPT	No prompt is generated: - In input processing mode, the terminal is switched to input mode when a H_GET primitive is issued without visible notification. - In output processing mode, the text is displayed without a prefix.
NMSG	All messages sent to the terminal by any agent other than the interactive step are held until the file is closed, or a H_TN primitive with the MSG option is issued for the terminal. If several files are simultaneously open for the terminal, all messages are held, if NMSG is selected for at least one of the files.
EOF	A character string, entered at the terminal, to simulate an End-of-File. This string may be prefixed by a $ and terminated by several spaces and a semicolon. The default EOF is the end of sequence, EOS. No other character can be entered on the line containing the EOF.

RECSIZE	The maximum record length. It is limited to 264 characters (prompt length and any expanded tabulation characters included). For a TN file, RECSIZE must be given. Note that the default is 0, and that a label cannot update this value.
BLKSIZE	Specifies the size of the block. The size must be at least equal to the size of the maximum record length. This parameter is not used by TAM, but the FD-mechanism requires it when the RECSIZE parameter is specified, otherwise a warning may occur at MACPROC time.
DATAFORM	Specifies the format of the data records: - SARF: Standard Access Data Format. Data only, with no header. SARF is the default format. - SSF: System Standard Format. Data records, and record header and control records. SSF can be used to present data as on a line printer with headings, footings, line numbering, margins, line spaces, etc.
REFMODE	Reference mode, either MOVE or LOCATE. See the *GPL System Primitives Reference Manual*. The default is MOVE mode. REFMODE can be modified by the reference mode option of H_OPEN primitive. LOCATE mode is possible only with Input processing mode (PMD parameter of the H_OPEN primitive).

4.6.2 H_OPEN (Open File)

Purpose:

H_OPEN is used to initialize the processing of a file, and must be issued before any other imperative can be used on a file. If a file is to be processed further, after an H_CLOSE has been issued, an H_OPEN must be reissued.

Syntax:

```
$H_OPEN

      ifn

      PMD = { INPUT | OUTPUT | APPEND | iv_char_2 }

      [ REFMODE = i_char_1 ] ;
```

Parameters:

ifn	Specifies the name of the file to be opened.
PMD	Specifies the processing mode, as follows: - INPUT makes the file accessible to H_GET, - OUTPUT makes the file accessible to H_PUT, - APPEND makes the file accessible to H_PUT, - iv_char_2 allows the user to dynamically change the processing mode. The variable, declared as CHAR(2), stands for one of three possible abbreviations: IN, OU, or AP.
REFMODE	Specifies the reference mode. Options are: - For MOVE mode, REFMODE ="M". - For LOCATE mode, REFMODE = "L".

4.6.3 H_GET (Get Record)

Purpose:

H_GET specifies the location and length of a record to be retrieved from a terminal file. It can be executed only in INPUT mode.

Syntax:

```
$H_GET

      ifn

      WA = { ov-location | ov-ptr }

      [ OUTLEN = ov_fb15 ]

      [ MAXLEN = i_fb15 ] ;
```

Parameters:

ifn	Specifies the name of the terminal file.
WA	Work area. Specifies the location of the record, as follows: - In MOVE mode, it is the area into which data received from the terminal is stored (ov_location). - In LOCATE mode, it is a pointer to the record in the buffer (ov_ptr).
OUTLEN	The length of the output field which is to receive the record (the length includes the SSF header if DATAFORM is SSF).
MAXLEN	The length, in bytes, of the work area. This parameter is only valid in MOVE reference mode. If the record obtained from the terminal is longer than the specified work area, it is truncated, and the normal return code, WALIM, is given. Processing can continue. The default is the actual length of the record.

Constraints:

If a function key has been pressed as part of the message to be received, two cases may occur:

- Either a text, previously assigned to this function key, is returned as a part of the message and the program is unaware that it came from a function key.
- Or no text has been assigned to this function key, in which case the first two characters of the work area are filled with the value of the function key and an appropriate normal return code is sent to the program (FUNCKEY).

In either case, if some text has been entered from the keyboard, it is appended to the text generated by the function key.

If the previous primitive issued for this ifn was H_TN with the TRSPRT parameter, the data obtained from the terminal is returned without alteration.

If the previous primitive issued for this ifn was H_TN with a temporary prompt, this text is sent to the terminal (less prompt) before the terminal is turned to input mode. The get input is then concatenated with the temporary prompt.

In transparent mode, if NPROMPT parameter in H_FD primitive, and if PROMPT parameter is not used in previous H_TN primitive, the terminal is just turned to input mode without any visible notification.

4.6.4 H_PUT (Put Record)

Purpose:

H_PUT is used to display a logical record. This command may be executed only in OUTPUT and APPEND processing modes.

Syntax:

```
$H_PUT

      ifn

      WA = iv_location

      [ LENGTH = i_fb15 ] ;
```

Parameters:

ifn	Specifies the internal file name.
WA	The user work area from which the record is to be moved.
LENGTH	The length, in bytes, of the next variable length record to be displayed. If the records are of fixed length, it is the length of the given string. This string may be extended with blanks to reach record size. The default length is record size.

Constraints:

The writing of the record may be deferred by TAM, but any H_GET primitive for a file that is assigned to the same terminal or a H_TN with the FORCEWRITE option forces the display of all unprocessed records.

4.6.5 H_CLOSE (Close File)

Purpose:

H_CLOSE is used to close a file.

Syntax:

```
$H_CLOSE

      ifn

      [ DEASSIGN ] ;
```

Parameters:

ifn	Specifies the name of the file.
DEASSIGN	The file is to be deassigned immediately, without waiting until the end of the step. The file cannot be reopened before it is reassigned.

4.6.6 H_TN (File declaration)

Purpose:

H_TN is used to dynamically redefine certain TAM options. This primitive is effective only when the file is open. If executed on a closed file, H_TN provokes an abnormal return code.

Syntax:

```
$H_TN

      ifn
                                                 { i_char1 }
      [ PROMPT = i_char12  LENGTH = i_fb15  MODE = { "T"     } ]
                                                 { "P"     }
      [ TRSPRT ]

      [ MSG = { "Y" | "N" | i_char1 }]

      [ FORCEWRITE ] ;
```

Parameters:

ifn	Specifies the internal file name assigned to the terminal. The file must currently be open.
PROMPT	Specifies the user work area which contains the value of the new prompt.
LENGTH	Specifies the length in bytes of the new prompt.
MODE	Specifies that prompt switching is either Permanent or Temporary: - P (Permanent): the prompt is switched for all subsequent H_PUT/H_GET primitives. The length of the prompt (LENGTH) must not exceed 12 characters. Setting LENGTH to 0 means no prompt. - T (Temporary): the prompt will apply only to the next H_GET primitive. In this mode, a prompt larger than 12 characters can be defined. The prompt length (LENGTH) must not exceed RECSIZE.
TRSPRT	The next H_PUT primitive will be transparent; i.e., data will be sent to the terminal without alteration. Note that when using TRSPRT, you must provide all the presentation characters. Presentation characters (CRLF, cursor position, etc.) will be treated as one line by the presentation layer to avoid malfunction.

MSG	Asynchronous messages are to be accepted (Y -yes) or deferred (N -no).
FORCEWRITE	The contents of the buffer associated with the terminal are sent to the terminal. It is only meaningful in OUTPUT or APPEND processing modes.

Constraints:

If this primitive is executed on a data management file, the abnormal return code FUNCNAV is returned and no action is taken.

4.7 OTHER LANGUAGES

The features of the standard programming languages (COBOL, FORTRAN) which handle access to a conventional sequential file also apply to the case where a terminal is being used instead. However, the terminal must be assigned via GCL.

For more information about such data management interfaces, refer to the appropriate programming language manual.

However, parameters specific to the terminal (see the H_TN primitive in Section 4) such as the temporary prompt and transparent mode are not available through these languages.
For example, to switch a terminal into transparent mode dynamically from a COBOL program, you must call a GPL procedure that uses the H_TN primitive.

An example of a COBOL program that reads from and writes to a terminal (until an end-of-file string, default value $EOS, is encountered), is given below.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     TERM-I-O.
*  THIS PROGRAM ILLUSTRATES THE USE OF A TERMINAL *
*  AS A SEQUENTIAL FILE FOR INPUT/OUTPUT *
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. LEVEL-64.
OBJECT-COMPUTER. LEVEL-64.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT INF ASSIGN INFILE ORGANIZATION SEQUENTIAL.
  SELECT OUTF ASSIGN OUTFILE ORGANIZATION SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD  INF
    LABEL RECORD STANDARD.
01  INF-REC PIC X(75).
FD  OUTF
    LABEL RECORD STANDARD.
01  OUTF-REC PIC X(75).
     .....
     .....
     .....

PROCEDURE DIVISION.
START-HERE.
    OPEN INPUT INF OUTPUT OUTF.
ITER.
    MOVE SPACES TO INF-REC
    READ INF AT END
                    CLOSE INF
                    CLOSE OUTF
                    STOP RUN.
    MOVE INF-REC TO OUTF-REC.
     .....
     .....
     .....
    WRITE OUTF-REC.
    GO TO ITER.
```

5. Forms Mode: SDPI Interface

5.1 FORM GENERATION

Forms are generated by the MAINTAIN_FORM (MNFORM) utility.

In EDIT mode, the form is described routine by routine in the SDPI form definition language. SDPI is short for Standard Device Programmatic Interface.

In FORMS mode, the form is described pictorially using the full screen terminals which support this mode, in conjunction with the form definition language.

For each form, MAINTAIN_FORM generates these members:

- A reference source member in a binary library (BINLIB). MAINTAIN FORM names this member, formname_SF.

- A terminal-independent object member (formname_of_ANY) or as many terminal dependent object members as there are terminal types for which the form has been oaded (formname of terminaltype). These members are created in a binary library BINLIB).

- Language members in a source library (SLLIB). These members, which contain data structures, are retrieved by the application using COPY routines or an equivalent, and include members for C Language or macro-instructions for GPL. Three types of members are generated:

  - formnameI defines the form identification. It identifies the form and indicates where to mount the form on the screen,

  - formnameV defines the selection vector. It is used to select variable fields in the form,

  - formnameR defines the data record. This data record is an image of the variable fields of the form.

The object forms are searched for in these binary libraries, according to standard search rules: (BLIB, BINLIB1, BINLIB2, BINLIB3).

You can optionally use a UFAS sequential file (OUTFILE) or a diskette (DK) to store terminal object members.

Access to forms is given to the application through the SDPI primitives. These primitives will call a forms runtime package which handles the presentation of forms, using the object form member created by MAINTAIN_FORM.

For more information about MAINTAIN_FORM, refer to the *GCOS7 Forms User's Guide*.

5.2 PRINCIPLES OF THE SDPI INTERFACE

As regards programs, access to forms mode is made through SDPI routines which make reference to language members. These routines provide a basic way to dialogue with the terminal.

Each routine results in an action being taken, except the CDFIDI routine which is purely informative. This action may result either in issuing some kind of message to the terminal (CDGET, CDSEND, CDATTR, CDATTL, CDMECH, CDRELS) or in receiving a message (CDRECV, CDPURGE).

For each action of the first kind, the program can specify an enclosure level in the routine, which is used to specify when the message is to be delivered to the terminal, and, if it is to be delivered immediately, whether the program retains the right to send other messages or gives control to the terminal.

The Receive routine (CDRECV) returns an enclosure level which tells the user or his program whether the message has been issued and whose turn it is to reply.

For each routine, a return code indicates whether it was successfully completed.

5.3 FORM ACTIVATION

Activating a form consists of displaying on the screen the fixed fields, their initial values, and the initial rendition attributes of the variable fields of a form. To activate a form, the program must issue a CDGET routine referencing a form identification structure that identifies the form and its mode of activation. Four activation modes are available: APPEND, OVERLAY, WINDOW, and ERASE.

In APPEND mode the user can specify where to mount the form on the screen, relative to forms already mounted. If a new form is activated, all the forms below the form to which the new form is appended are implicitly released and the area occupied by these forms on the screen is cleared. If a form is requested to be mounted at the top of the screen, the whole screen is cleared. Several occurrences of the same form can be mounted simultaneously on the screen. In this case, each occurrence is identified by an occurrence number that is specified in formnameI and formnameV structures.

In OVERLAY mode all the preceding forms are logically released but the screen is not cleared. These forms become frozen and the new form is displayed on the screen without relocation.

The WINDOW mode is a mode in which each form defines a window on the screen. The window consists of the smallest rectangle that may contain the form beginning in line 1 column 1 up to the maximum line and the maximum column.

The program specifies, in the form identification structure, the location of the window by giving the coordinates of its top left corner. All the forms previously on the screen are frozen, as in OVERLAY mode. Then the contents of the window that will contain the form are erased. The form is relocated so as to be placed in the window and it becomes the new active form.

The main characteristic of the WINDOW mode is that when the CDGET references a form which is already displayed in a window, this form becomes the active form again with its fields restored to their contents when the form was the active form. In this case, the form is always displayed at the same location on the screen and the window coordinates that may be specified are ignored. This gives a visibility similar to a stack of sheets of paper that partially overlay each other and where the user may remove a sheet in the middle of the stack to put it on the top of the stack.

Also, the POPUP mechanism may be used to release only the currently active form. In this case, the form window is reset to its underlaying contents and the next form in the stack becomes the new active form.

The ERASE mode is similar to the WINDOW mode, with the difference that it first releases all forms and clears the screen.

A form may be activated in WINDOW mode only if the previous form was activated in one of these modes: ERASE, WINDOW. A form cannot be appended to a form activated in WINDOW or ERASE mode.

The object form may be mounted either from the binary libraries of the step or from the terminal diskette. This choice is transparent to the program which always issues the same routine. The forms runtime package either sends the object form or requests the terminal to mount the form from the diskette, depending on the form generation options.

NOTES:

1. The TERM=ANY option is mandatory for a form to be activated in WINDOW or ERASE mode. (See the *GCOS7 Forms User's Guide*.)

2. Activating a form within an IOF startup procedure is not recommended.

5.4 OUTPUT ACTIONS

Several routines are available that display a message on the terminal:

- CDSEND is used to select variable fields of a form. A selection vector is passed to identify the form and to specify the fields that must be modified.
- CDATTR is used to define a rendition attribute of fields of a form. The form and its fields are identified by a selection vector.
- CDATTL is similar to CDATTR except that it sets a list of attributes instead of only one attribute.
- CDMECH is a control function that acts on the terminal, clearing all unprotected fields or setting off the alarm.
- CDRELS releases all the active forms and sets the terminal to line mode. This routine does not clear the screen. If the screen is to be cleared, this routine must be followed by a CDMECH with RESET option.

5.5 INPUT ACTIONS

When the terminal user enters data on the screen, his program must issue as many CDRECV routines as there are forms for which data is to be entered. It is recommended that each CDRECV be preceded by a CDFIDI routine which returns the identification of the next form to receive data. The program passes a selection vector to the CDRECV routine to identify this form and to select those fields of the form to receive data.

If a function key has been pressed and a function key field has been implicitly or explicitly declared at form generation time, this field, if selected, is filled with the rank of the function key. The mapping of function keys onto terminal-independent ranks is detailed in the *GCOS7 Forms User's Guide* .

If the message contains information relative to another form, the program is notified through an appropriate enclosure level. If the program does not want to receive the pending data, it must issue a CDPURGE routine.

5.6 TRACES

The FORMS runtime package provides traces to debug user programs and to serve as a diagnostic tool in case of problems. These traces are activated through the LOG directive under IOF.

They consist of:

- A trace of each user routine. This trace is issued after each routine is processed; it displays all the parameters of the routine after its execution (the selection vector and the data record contents when used), as well as the routine's return code.
- A trace of all messages received and sent. These messages are designated as INBOUND and OUTBOUND. They are traced as they are processed by FORMS. Therefore, the traces show the anticipation of input message handling.
- Data structures. These are the Data Map, which contains internal mapping information generated by MAINTAIN_FORM, and the terminal independent Preform structure for dynamic forms.

The last two traces are activated by using the option LOG MODIFY ON=*F or **, or LOG START FOR = SYSTEM.

5.7 SDPI COBOL STATEMENTS

This section describes the FORMS mode routines accessible via COBOL statements. For the C Language interface see the manual *C Language System Primitives*, and for the GPL interface see *GPL System Primitives*.

Some parameters are passed to the input or output CD structure.

Since the CD cannot be directly used in a COBOL CALL routine, it must be redefined in the Communication section of an I/O structure.

Such a redefinition is subsequently designated as a "CD" alias.

5.7.1 Data Structures

The FORMS services accept several data structures as parameters. The form identification, the data record and the selection vector are generated by MAINTAIN_FORM and retrieved by COPY in the program.

The input and output Communication descriptions are standard COBOL communication structures. The actual use of these structures by FORMS is detailed below.

The input and output Communication Description (CD) entries must be redefined in the Communication Section, because the CD names cannot be used directly in the CALL verb's parameter list. The redefinitions are:

```
CD cd-name-1 [FOR] INPUT.
01 cd-alias-1.
  02 symbolic-queue-data-name    PIC X(12).   (c)
  02 FILLER                      PIC X(36).   (a)
  02 date-data-name              PIC X(6).    (e)
  02 time-data-name              PIC X(8).    (e)
  02 symbolic-source-data-name   PIC X(12).   (b)
  02 text-length-data-name       PIC 9(4).    (b)
  02 end-key-data-name           PIC X.       (f)
  02 status-key-data-name        PIC XX.      (g)
```

and:

```
CD cd-name-2 [FOR] OUTPUT.
01 cd-alias-2.
  02 destination-count-data-name     PIC 9(4) VALUE 1.   (h)
  02 text-length-data-name           PIC 9(4).    (b)
  02 status-key-data-name            PIC XX.      (g)
  02 error-key-data-name             PIC X.       (b)
  02 symbolic-destination-data-name  PIC X(12).   (d)
```

NOTES:

a) The FILLERs in the description are not used.

b) These fields are not used under IOF.

c) This field must contain the string "CONSOLE" when calling any FORMS service using the input CD.

d) This field must contain the string "CONSOLE" when calling any FORMS service using the output CD.

e) After a CDRECV service, these fields contain the date and the time of day.

f) After a CDRECV service, this field returns the enclosure level associated with the received message. It may be either of the following:

"1":further data, coming from another form, is still to be incorporated in the message.

"3": all data has been received. Control is given to the program.

g) After any FORMS service, this field contains the status on completion of the statement. The various status keys are detailed in the following routine descriptions.

h) This field must be set to 1.

5.7.2 CDSEND (Forms Send)

Purpose:

This routine causes data in a data record to be transferred to the endpoint indicated by the selection vector (that is, a logical terminal as seen by the program).

Syntax:

```
CALL "CDSEND" USING output-CD-alias,
        data-record, enclosure-level, selection vector.
```

Parameters:

output-CD-alias	Refer to the CD description.
data record	The data record associated with the form. This structure is output by MAINTAIN_FORM (formnameR).
enclosure-level	The enclosure level to be associated with the data. This is a one character data item that may take these values: "1" End of record: Data is quarantined; no message is sent to the terminal. "2" End of quarantine: The data relative to this command and all previously quarantined data are sent to the terminal. The application keeps control. "3" End of interaction: The data relative to this command and all previously quarantined data are sent to the terminal. Control is given to the terminal.
selection-vector	The selection vector. This structure is output by MAINTAIN_FORM (formnameV). The occurrence number field of the selection vector must identify the form occurrence to which the command applies. The other fields of the selection vector must be loaded as follows: - "space": do not move the associated data field from the data record. - "S": move the associated data field from the data record. - "C": clear the contents of the field. After execution, the contents of the selection vector fields are unchanged , except in case of a selected numeric field with invalid contents where the selection vector field is loaded with "A".

Return Codes:

Normal:

0 DONE

AB ALMOST:	an error was found in at least one field
AF DONEIDE:	invalid selection vector contents

Abnormal:

9A BREAK:	a break occurred
AC ARGERR:	unexpected parameter
A0 SNDVIOL:	turn error
A7 SNDARERR:	the selection vector does not match an active form
A1 SEQERR:	endpoint not in FORMS mode
97 OPTERR:	invalid enclosure level
AG NOMATCH:	the creation date of the form does not match the creation date of program structures
A9 DVIDFBID:	device not supported
9H DNSPEC:	invalid endpoint name
S1 DAMAGED:	system error
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents

5.7.3 CDRECV (Forms Receive)

Purpose:

This routine causes data to be received in the data record according to the selection vector.

Syntax:

```
CALL "CDRECV" USING input-CD-alias,
          data-record, wait-indicator, selection-vector.
```

Parameters:

input-CD-alias	Refer to CD description.
data-record	The data record associated with the form. This structure is output by MAINTAIN_FORM (formnameR).
selection-vector	This structure is output by MAINTAIN FORM (formnameV). The form name and occurrence number fields of the selection vector must match a form and occurrence number for which data is available. For each field in which a value is to be received from the terminal, the corresponding selection vector field must be loaded with "S". Such a selection vector field will be modified by the command as follows:

R	valid data has been moved to the corresponding field.
S	no data available for the field; i.e., either the field is not a transmittable field or its contents are set to null or to spaces and the RECEIVE SPACES option was not selected for the form.
C	this field contains the cursor (may also be returned for a non-selected field).
X	this field contains the cursor and valid data has been moved to it (R + C).
D	this field has the attribute DT or IT and data are received in the field.
+	a sign was entered in an unsigned field, no data transferred.
T	least significant digits have been truncated, data transferred.
O	significant digits would have been truncated, no data transferred.
A	incorrect characters input to the field, no data transferred.

O, T and + may be returned only for numeric or numeric edited fields; that is, fields with a numeric or numeric edited screen picture or with a computational usage (e.g. COMP.1 fields).

All other selection vector fields must be loaded with spaces. If some data is available for a non selected field, the data is not transferred and the corresponding selection vector field is set to L (=lost), otherwise the selection vector field is unaffected.

wait-indicator	one character to be set to "0" (provision for future use).

Return Codes:

Normal:

0 DONE

AF DONEIDE:	invalid selection vector contents
AB ALMOST:	at least one field in error
A8 SKIPPED:	some received fields were not selected; they are lost
AI IGNORED:	unexpected receive function key

Abnormal:

9A BREAK:	a break occurred
AC ARGERR:	unexpected parameter
A3 RCVVIOL:	turn error
A7 RECARERR:	the selection vector does not match an active form
A1 SEQERR:	endpoint not in FORMS mode
AG NOMATCH:	the creation date of the form does not match the creation date of program structures
A9 DVIDFBID:	device not supported
9H DNSPEC:	invalid endpoint name
S1 DAMAGED:	system error
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents

5.7.4 CDGET (Form Activation)

Purpose:

This routine causes a form to be activated.

Syntax:

```
CALL "CDGET" USING output-CD-alias,
             format [,enclosure-level].
```

Parameters:

output-CD-alias — Refer to CD description.

format — A structure that identifies the form to be activated and where it has to be mounted on the screen. This structure is output by MAINTAIN FORM (formnameI) and is used as follows:

- If APPEND mode is specified (i.e., the mode field of the format structure is set to A), all forms following the form specified by the old form name and occurrence number fields of the format structure are released and cleared. The new form is appended after the old form. If the old form and occurrence number fields specify a name consisting of spaces and an occurrence of zero (default initialization), the new form will be appended to the top of the screen and all other forms are released and cleared.

- If OVERLAY mode is specified (i.e., the mode field of the format structure is set to O), the old form and occurrence number fields must specify a name consisting of spaces and an occurrence number of zero. The forms that were active are frozen; i.e., they remain visible on the screen but all their fields become protected and they are no longer addressable from the program. The new form is activated at the top of the screen. There is no checking of overlapping of the newly activated form with other forms.

- If WINDOW mode is specified, that is if the field "form-name-MD" is set to "W", the forms that were active are frozen. If the form to be activated is already displayed on the screen with the same occurrence number, this form is put on top of other forms and made active again with its previous contents. Otherwise the "form-name-SL" and "form-name-SC" fields determine the line and column numbers of the top left corner of the rectangle where the form is to be placed. The contents of this rectangle are cleared and the form is made active.

- If ERASE mode is specified, that is if the field "form-name-MD" is set to "E", all the forms are released and the screen is cleared. The "form-name-SL" and "form-name-SC" fields determine the line and column numbers of the top left corner of the rectangle where the form is to be placed.

If a CDGET with OVERLAY is followed by one or more CDGETs with APPEND, the subsequent forms will be appended to the overlaying form.

The formname-LL field of the format structure gives the number of lines allocated to the new form. If formname-LL is zero, the number of lines allocated is equal to the number of lines in the form. If formname-LL is not zero and is greater than or equal to the number of lines in the form, the form is completed with the appropriate number of blank lines. If formname-LL is not zero and is less than the number of lines in the form, the status FUNCNAV is returned. This field is interpreted only for the new format structure, that is, the first FILLER must be set to 2.

The maximum number of active or frozen forms is 6.

enclosure-level	The enclosure level to be associated with the data. This is a one-character data item that may take these values: "1" End of record: Data is quarantined; no message is sent to the terminal. "2" End of quarantine: The data relative to this command and all previously quarantined data are sent to the terminal. The application keeps control. "3" End of interaction: The data relative to this command and all previously quarantined data are sent to the terminal. Control is given to the terminal. The default is "1".

Return Codes:

Normal:

0 DONE

Abnormal:

9A BREAK:	a break occurred
9J ALREADY:	form already active (last form activated in WINDOW mode)
AC ARGERR:	unexpected parameter
A0 SNDVIOL:	turn error
A4 RECNFD:	the specified form was not found
AE FUNCNAV:	activation mode not available
97 OPTERR:	invalid enclosure level
92 ENTRYOV:	Maximum number of active or frozen forms exceeded
or MSGOV:	expansion area overflow
or SPACEOV:	no more space available for control structures
A9 DVIDFBID:	device not supported
AG NOMATCH:	the creation date of the form does not match the creation date of program structures
9H DNSPEC:	invalid endpoint name
9F CALLVIOL:	environment is neither TDS nor IOF
S1 DAMAGED:	system error
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents

5.7.5 CDRELS (Release all forms)

Purpose:

This routine releases all active forms and puts the terminal in line mode. The screen is not cleared. This routine also releases all attribute modifications into the forms management structure. No attribute modification can be removed by another CALL after this routine has been performed.

Syntax:

```
CALL "CDRELS" USING output-CD-alias [,enclosure-level].
```

Parameters:

output-CD-alias	Refer to CD description.
enclosure-level	The enclosure level to be associated with the data. This is a one character data item that may take these values: "1" End of record: Data is quarantined; no message is sent to the terminal. "2" End of quarantine: The data relative to this command and all previously quarantined data are sent to the terminal. The application keeps control. "3" End of interaction: The data relative to this command and all previously quarantined data are sent to the terminal. Control is given to the terminal. The default is "1".

Return Codes:

Normal:

0 DONE

Abnormal:

9A (BREAK):	a break occurred
AC ARGERR:	unexpected parameter
A0 SNDVIOL:	turn error
A1 SEQERR:	endpoint not in FORMS mode
97 OPTERR:	invalid enclosure level
A9 DVIDFBID:	device not supported
9H DNSPEC:	invalid endpoint name

92 MSGOV:	expansion area overflow
S1 DAMAGED:	system error
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents

NOTE: After calling CDRELS, the released forms are still displayed. Activation of a new form by CALL CDGET in OVERLAY or WINDOW mode could lead to unpredictable results.

5.7.6 CDPURGE (Purge input data)

Purpose:

This routine purges all pending input messages and gives control back to the application.

Syntax:

```
CALL "CDPURGE" USING input-CD-alias.
```

Parameters:

input-CD-alias	Refer to CD description.

Return Codes:

Normal:

0 DONE

Abnormal:

9A BREAK:	a break occurred
A1 SEQERR:	endpoint not in FORMS mode
9H DNSPEC:	invalid endpoint name
S1 DAMAGED:	system error
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents

5.7.7 CDATTR (Attribute selection)

Purpose:

This routine causes an attribute to apply to the fields selected in the selection vector. This routine cannot remove an attribute modification performed before a CALL CDRELS.

Syntax:

```
CALL "CDATTR" USING output-CD-alias, selection-vector,
                    attribute-identifier, [enclosure-level].
```

Parameters:

output-CD-alias	Refer to CD description.
selection-vector	The selection vector. This structure is output by MAINTAIN_FORM (formnameV). The form name and occurrence number fields of the selection vector must identify the form occurrence to which the command applies. The other fields of the selection vector must be loaded as follows: "space": do not assign the attributes of the associated field. "S": assign the attributes of the associated field. "C": clear the contents of the field. "B": both clear the contents of the field and assign the attributes as specified. After execution, the contents of the selection vector fields are unchanged.
attribute identifier	The attribute to be applied to the fields selected. This is a four character alphanumeric data item that may take these values: BI: Blink BD: Bold (i.e., high intensity). Bxxx: Background color where xxx may be: RED: Red YEL: Yellow BLU: Blue GRE: Green CYA: Cyan MAG: Magenta WHI: White BLA: Black DFT: Default color CN: Conceal. COS: Column Separator. CP: Cursor position. DFT: Default rendition (i.e., NBI, NHL, NRV, NCOS, NUL, BDFT, FDFT, and normal intensity). FT: Faint (i.e., reduced intensity). Fxxx: Foreground color (xxx may take the same values as for Bxxx).

	HL: Rendition highlighted in a terminal-specific manner. INIT: Initial attributes. NBI: No blink NCOS: No column separator NHL: Not highlighted. NPR: Not protected. NRV: No reverse video. NTR: Not transmittable. NUL: Not underlined. PR: Protected. RV: Reverse video. TR: Transmittable. UL: Underlined. Once an attribute is modified, it remains modified until either another CDATTR or CDATTL command specifies a contradictory attribute for the same field or a CDMECH command with INITAT argument is executed. When a CP attribute is applied to a protected field the cursor cannot be positioned on that field, so it will be positioned either on the first field of the screen to which the CP attribute was given at form generation (if any), or on the first non-protected field of the screen (home field).
enclosure-level	The enclosure level to be associated with the data. This is a one character data item that may take these values: "1" End of record: Data is quarantined; no message is sent to the terminal. "2" End of quarantine: The data relative to this command and all previously quarantined data are sent to the terminal. The application keeps control. "3" End of interaction: The data relative to this command and all previously quarantined data are sent to the terminal. Control is given to the terminal. The default is "1".

Return Codes:

Normal:

0 DONE	
AF DONEIDE:	invalid selection vector contents.

Abnormal:

9A BREAK:	a break occurred
AC ARGERR:	unexpected parameter
A0 SNDVIOL:	turn error
A7 SNDARERR:	the selection vector does not match an active form
A1 SEQERR:	endpoint not in FORMS mode
97 OPTERR:	invalid enclosure level
A6 OBJUNKN:	unknown attribute
AG NOMATCH:	the creation date of the form does not match the creation date of program structures
A9 DVIDFBID:	device not supported
9H DNSPEC:	invalid endpoint name
or CONFLICT:	the attribute conflicts with the form (e.g., because the selected field starts in column 1)
92 MSGOV:	expansion area overflow
S1 DAMAGED:	system error
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents

5.7.8 CDATTL (Attribute list selection)

Purpose:

This routine causes a list of attributes to apply to the fields selected in the selection vector. Attributes are evaluated sequentially, hence the effect of one attribute may be cancelled by the following attributes in the list.

Syntax:

```
CALL "CDATTL" USING output-CD-alias, selection vector,
      attribute-identifier, [enclosure-level].
```

Parameters:

output-CD-alias

Refer to CD description.

selection-vector

This structure is output by MAINTAIN_FORM (formnameV). The form name and occurrence number fields of the selection vector must identify the form occurrence to which the command applies. The other fields of the selection vector must be loaded as follows:

"space": do not assign the attributes of the associated field.
"S": assign the attributes of the associated field.
"C": clear the contents of the field.
"B": both clear the contents of the field and assign the attributes as specified.

After execution, the contents of the selection vector fields are unchanged.

attribute-identifier

A structure with this description:

```
01 data-name1.
02 data-name2 PIC 9(3) VALUE n.
02 data-name3 PIC X(4) OCCURS n.
```

where data-name2 is the number of attributes to be applied coded in decimal and each element of data-name3 is an attribute identifier. The list of attribute identifiers and their interpretations are detailed in the description of CDATTR command.

enclosure-level

The enclosure level to be associated with the data. This is a one-character data item that may take these values:

"1" End of record: Data is quarantined; no message is sent to the terminal.

"2" End of quarantine: The data relative to this command and all previously quarantined data are sent to the terminal. The application keeps control.

"3" End of interaction: The data relative to this command and all previously quarantined data are sent to the terminal. Control is given to the terminal.

The default is "1".

Return Codes:

Normal:

0 DONE	
AF DONEIDE	invalid selection vector contents

Abnormal:

9A BREAK	a break occurred
AC ARGERR	unexpected parameter
A0 SNDVIOL	turn error
A7 SNDARERR	the selection vector does not match an active form
A1 SEQERR	endpoint not in FORMS mode
97 OPTERR	invalid enclosure level
A6 OBJUNKN	unknown attribute
AG NOMATCH	the creation date of the form does not match the creation date of program structures
A9 DVIDFBID	device not supported
or CONFLICT	the attribute conflicts with the form (e.g., because the selected field starts in column 1)
9H DNSPEC	invalid endpoint name
92 MSGOV	expansion area overflow
S1 DAMAGED	system error
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents

5.7.9 CDFIDI (Form identification)

Purpose:

This routine causes the system to return the name of the form relative to the next message.

Syntax:

```
CALL "CDFIDI" USING input-CD-alias, form-identifier.
```

Parameters:

input-CD-alias	Refer to CD description.
form identifier	The name and occurrence of the form relative to the next message (or portion of message) to be received. The format of this structure is as follows: 01 form-identification. 02 form-name PIC X(8). 02 occurrence-number PIC 9(3).

Return Codes:

Normal:

0 DONE

Abnormal:

A3 RCVVIOL:	turn error
9A BREAK:	a break occurred
A9 DVIDFBID:	device not supported
9H DNSPEC:	invalid endpoint name
S1 DAMAGED:	system error
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents

5.7.10 CDMECH (Control function mechanism)

Purpose:

This statement activates a control function mechanism. Note that it cannot remove an attribute modification performed before a CALL CDRELS.

Syntax:

```
CALL "CDMECH" USING output-CD-alias, mechanism-identifier
[,enclosure-level].
```

Parameters:

output-CD-alias	Refer to CD description.
mechanism-identifier	A mnemonic that represents a control function mechanism that is related to the device. This is a 6-character alphanumeric data item that may take these values:
ALARM:	activate the audio or visual alarm, if any.
CLEAR:	clear all unprotected fields.
PROTCT:	turn to protected all named fields of all active forms. This command is effective on the next CDGET.
INITAT (alias RESET):	clear unprotected fields and reset attributes to their initial value. When the screen is in normal mode, (i.e., after CDRELS LEVEL=1), this command clears the screen.
INIT:	resets all forms to their initial state.
STPRA and STPRV:	are valid only if the endpoint is a forms mode printer, otherwise they have no effect. If the STPRV mechanism is set, then for all subsequent CDSEND commands, only the variable fields are printed. If the STPRA mechanism is set, then for all subsequent CDSEND commands, all fields are printed. STPRA is the default mode.
CPON:	sets up a mode where the user will be notified of the cursor position in the subsequent CDREV statements, for terminals that support this feature (QUESTAR 200, IBM3278/3279, MINITEL).
CPOFF:	resets a previously CPON mechanism.
POPUP:	When the active form has been activated in window mode, this mechanism causes this form to be released and the previous form in the stack of displayed forms to become active. The window associated with the released form is reset to the underlying contents. When the active form has not been activated in WINDOW mode, or when the stack of displayed forms is reduced to one form, a return code FUNCNAV is sent.

enclosure-level	The enclosure level to be associated with the data. This is a one character data item that may take these values: "1" End of record: Data is quarantined; no message is sent to the terminal. "2" End of quarantine: The data relative to this command and all previously quarantined data are sent to the terminal. The application keeps control. "3" End of interaction: The data relative to this command and all previously quarantined data are sent to the terminal. Control is given to the terminal. The default is "1".

Return Codes:

Normal:

0 DONE

Abnormal:

9A BREAK:	a break occurred
AC ARGERR:	unexpected parameter
A0 SNDVIOL:	turn error
97 OPTERR:	invalid enclosure level
A6 OBJUNKN:	unknown mechanism
A9 DVIDFBID:	device not supported
9H DNSPEC:	invalid endpoint name
S1 WRONGITM:	invalid object form contents
S1 WRONGSTA:	invalid data map contents
AE FUNCNAV:	mechanism not available

5.8 A PROGRAMMING EXAMPLE

This is an example of a COBOL program using a form under IOF. The form is called SALES and is used for recording the sales of representatives. The program calculates the total sales value for each product and a grand total for the sales of all products. If data is entered incorrectly, the fields of the corresponding line are highlighted in a terminal-dependent way and the data can be re-entered.

The form image of SALES is as follows:

```
            # PRODUCT TYPE ##########
NAME OF REPRESENTATIVE ########################### CODE #####
#
PRODUCT CODE QUANTITY  PRICE    TOTAL
######  ####   ###    ######   #######
######  ####   ###    ######   #######
######  ####   ###    ######   #######
######  ####   ###    ######   #######
######  ####   ###    ######   #######
######  ####   ###    ######   #######
######  ####   ###    ######   #######
GRAND TOTAL ##########
```

The corresponding form definition language is:

```
NF SLASH LINE IS 01 COL IS 02 SCREEN-PIC IS X(1) UL.
NF SALE-TITLE LINE IS 01 COL IS 37 SCREEN-PIC IS X(10) UL.
NF REP-NAME LINE IS 02 COL IS 29 SCREEN-PIC IS X(27) UL.
NF REP-CODE LINE IS 02 COL IS 64 SCREEN-PIC IS X(6) UL DI.
ARRAY SALE-LINES OCCURS 07.
NF PRODUCT LINE IS 04 COL IS 03 SCREEN-PIC IS X(6).
NF PRODUCT-CODE LINE IS 04 COL IS 18 SCREEN-PIC IS X(4) DI.
NF QUANTITY LINE IS 04 COL IS 35 SCREEN-PIC IS 9(3) DI.
NF PRICE LINE IS 04 COL IS 46 SCREEN-PIC IS ZZ9.99 NU.
NF TOTAL LINE IS 04 COL IS 59 SCREEN-PIC IS ZZZ9.99 PR.
UF LINE IS 04 COL IS 13 VALUE IS ":".
UF LINE IS 04 COL IS 27 VALUE IS ":".
UF LINE IS 04 COL IS 42 VALUE IS ":".
UF LINE IS 04 COL IS 55 VALUE IS ":".
END-ARRAY.
NF GRAND-TOTAL LINE IS 15 COL IS 50 SCREEN-PIC IS ******9.99 PR.
UF LINE IS 01 COL IS 23 VALUE IS "PRODUCT TYPE" RV.
UF LINE IS 02 COL IS 04 VALUE IS "NAME".
UF LINE IS 02 COL IS 09 VALUE IS "OF".
UF LINE IS 02 COL IS 12 VALUE IS "REPRESENTATIVE".
UF LINE IS 02 COL IS 58 VALUE IS "CODE".
UF LINE IS 03 COL IS 03 VALUE IS "PRODUCT".
UF LINE IS 03 COL IS 13 VALUE IS ":".
UF LINE IS 03 COL IS 18 VALUE IS "CODE".
UF LINE IS 03 COL IS 27 VALUE IS ":".
UF LINE IS 03 COL IS 31 VALUE IS "QUANTITY".
UF LINE IS 03 COL IS 42 VALUE IS ":".
UF LINE IS 03 COL IS 46 VALUE IS "PRICE".
UF LINE IS 03 COL IS 55 VALUE IS ":".
UF LINE IS 03 COL IS 60 VALUE IS "TOTAL".
UF LINE IS 15 COL IS 36 VALUE IS "GRAND".
UF LINE IS 15 COL IS 42 VALUE IS "TOTAL".
```

These data structures are generated:

SALESI:

```
*
02 FILLER    PIC X VALUE "3".
02 FILLER    PIC X(8) VALUE "SALES".
02 SALES-NO  PIC 9(3) VALUE ZERO.
02 SALES-MD  PIC X VALUE "A".
02 SALES-OF  PIC X(8) VALUE SPACES.
02 SALES-OO  PIC 9(3) VALUE ZERO.
02 SALES-LL  PIC 9(3) VALUE ZERO.
02 FILLER    PIC X(5) VALUE "87097".
02 FILLER    COMP-1 VALUE 01385.
02 SALES-AF  PIC 9 VALUE 1.
02 SALES-SL  PIC 9(3) VALUE 1.
02 SALES-SC  PIC 9(3) VALUE 1.
```

SALESR:

```
04 SALES-FC  PIC 99.
04 SLASH  PIC X(1).
04 SALE-TITLE  PIC X(10).
04 REP-NAME  PIC X(27).
04 REP-CODE  PIC X(6).
04 SALE-LINES-A .
     05 SALE-LINES OCCURS 7.
          06 PRODUCT  PIC X(6).
          06 PRODUCT-CODE  PIC X(4).
          06 QUANTITY  PIC 9(3).
          06 PRICE  PIC 999V99.
          06 TOTAL  PIC 9999V99.
04 GRAND-TOTAL  PIC 9999999V99.
```

SALESV:

```
02 SALESV.
03 FILLER  PIC X VALUE ""2"".
03 FILLER  COMP-1 VALUE 41.
03 FILLER  PIC X(8) VALUE "SALES".
03 SALES-VO  PIC 9(3) VALUE ZERO.
03 SALES-V.
     04 SALES-FC-V  PIC X.
     04 SLASH-V  PIC X.
     04 SALE-TITLE-VPIC X.
     04 REP-NAME-V  PIC X.
     04 REP-CODE-V  PIC X.
     04 SALE-LINES-AV .
          05 SALE-LINES-V OCCURS 7.
               06 PRODUCT-V  PIC X.
               06 PRODUCT-CODE-V  PIC X.
               06 QUANTITY-V  PIC X.
               06 PRICE-V  PIC X.
               06 TOTAL-V  PIC X.
04 GRAND-TOTAL-V   PIC X.
```

The program, which is called DEMO, is listed below:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMO.
AUTHOR. A N OTHER.
INSTALLATION. BULL CENTER.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. LEVEL-64.
OBJECT-COMPUTER. LEVEL-64.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 I  PIC 999.
77 J  PIC 9.
77 W-I  PIC 9.
77 ERROR-INDICATOR  PIC 9.
77 W-NBF  PIC 9.
77 W-ATTRIB  PIC  X(4)  VALUE  "INIT".
01 SALES-SW.
   COPY SALESV REPLACING TRAILING "-V" BY "-W".
01 ATTRIBUTE.
   02 FILLER  PIC 999 VALUE 002.
   02 FILLER  PIC X(4) VALUE "CP".
   02 FILLER  PIC X(4) VALUE "HL".
77 TOTAL-MAN  PIC 9(4)V99.
77 TOTAL-NUM  PIC 9(7)V99.
77 W-ATTRIBUT  PIC X(6).
77 W-LEVEL  PIC X.
01 MESS-AREA.
   02 FILLER  PIC X(8).
01 SALESI.
   COPY SALESI.
01 SALES-SV.
   COPY SALESV.
01 SALESR.
   COPY SALESR.
COMMUNICATION SECTION.
CD IQ INPUT
   QUEUE         ISQ
   MESSAGE DATE  IMD
   MESSAGE TIME  IMT
   SOURCE        ISS
   TEXT LENGTH   ITL
   END KEY       IEK
   STATUS KEY    ISK
   COUNT         IMC.
01     RIQ.
02     RISQ  PIC X(12).
02     RISU  PIC X(36).
02     RIMD  PIC 9(6).
02     RISS  PIC X(12).
02     RITL  PIC 9(4).
02     RIEK  PIC X.
02     RISK  PIC XX.
02     RIMC  PIC 9(6).
CD OQ OUTPUT
   DESTINATION COUNT     ODC
   TEXT LENGTH           OTL
   STATUS KEY            OSK
   ERROR KEY             OEK
   DESTINATION           OSD.
01  ROQ.
  02   RODC  PIC 9(4).
  02   ROTL  PIC 9(4).
  02   ROSK  PIC XX.
  02   ROEK  PIC X.
  02   ROSD.
    03 RNET  PIC X(4).
    03 RTRM  PIC X(4).
    03 RTYP  PIC X(3).
    03 RNUM  PIC X.
PROCEDURE DIVISION.
PROG-STAR.
     MOVE "CONSOLE" TO  ISQ OSD.
     MOVE 1 TO  ODC.
     MOVE 1 TO  OTL.
     MOVE "3" TO W-LEVEL.
     MOVE ALL "S" TO SALES-W.
     MOVE ALL SPACES TO SALESR.
CALL "CDGET" USING ROQ SALESI W-LEVEL.
IF OSK = "A4" GO TO END-PROG.
MAIN-LOOP.
     MOVE ALL SPACES TO SALESR.
     MOVE ALL "S" TO SALES-V.
     CALL "CDRECV" USING RIQ SALESR W-LEVEL SALES-SV.
     IF SLASH-V = "R" AND SLASH = "/"
        GO TO END-PROG.
MOVE ALL SPACES TO    SALE-TITLE-V
                      SLASH-V
                      REP-CODE-V
                      REP-NAME-V.
MOVE 0 TO ERROR-INDICATOR.
MOVE ZERO TO TOTAL-NUM.
PERFORM TOTAL-D THRU TOTAL-F VARYING I FROM 1 BY 1
                             UNTIL I > 7.
IF ERROR-INDICATOR = 1
   MOVE "1" TO W-LEVEL
   CALL "CDATTR" USING ROQ SALES-SW W-ATTRIB W-LEVEL
   MOVE "3" TO W-LEVEL
   MOVE SPACES TO GRAND-TOTAL-V
   CALL "CDATTL" USING ROQ SALES-SV ATTRIBUTE W-LEVEL
   MOVE SALES-V TO SALES-W
   GO TO MAIN-LOOP
ELSE
   MOVE "1" TO W-LEVEL
   MOVE ALL "S" TO SALES-W
   CALL "CDATTR" USING ROQ SALES-SW W-ATTRIB W-LEVEL
   PERFORM PREP-SEND THRU END-PREP-SEND VARYING J
      FROM 1 BY 1 UNTIL J > 7
   MOVE "S" TO GRAND-TOTAL-V
   MOVE TOTAL-NUM TO GRAND-TOTAL
   MOVE "3" TO W-LEVEL
   CALL "CDSEND" USING ROQ SALESR W-LEVEL SALES-SV
   MOVE "S" TO SLASH-V
   CALL "CDRECV" USING RIQ SALESR W-LEVEL SALES-SV.
IF SLASH-V = "R" AND SLASH = "/"
          GO TO END-PROG.
MOVE "1" TO W-LEVEL.
CALL "CDRELS" USING ROQ W-LEVEL.
MOVE ALL "S" TO SALES-V.
MOVE ALL SPACES TO SALESR.
MOVE "3" TO W-LEVEL.
CALL "CDGET" USING ROQ SALESI W-LEVEL.
GO TO MAIN-LOOP.
END-PROG.
    STOP RUN.
TOTAL-D.
     IF PRODUCT-V (I) NOT = "R" GO TO TOTAL-NR.
     IF PRODUCT-CODE-V (I) = "R"
     AND QUANTITY-V (I) = "R"
     AND PRICE-V (I) = "R"
         MULTIPLY QUANTITY (I) BY PRICE (I)
                               GIVING TOTAL-MAN
        MOVE TOTAL-MAN TO TOTAL (I)
        ADD TOTAL-MAN TO TOTAL-NUM
        MOVE ALL SPACES TO SALE-LINES-V (I)
        GO TO TOTAL-F.
TOTAL-NR.
        IF PRODUCT-CODE-V (I) = "S"
        AND QUANTITY-V (I) = "S"
        AND PRICE-V (I) = "S"
           MOVE ALL SPACES TO SALE-LINES-V (I)
           GO TO TOTAL-F.
TOTAL-ERROR.
     MOVE 1 TO ERROR-INDICATOR.
     MOVE ALL "S" TO SALE-LINES-V (I).
TOTAL-F.
    EXIT.
PREP-SEND.
     IF SALE-LINES (J) = SPACES
        MOVE ALL SPACES TO SALE-LINES-V (J)
ELSE
       MOVE ALL "S" TO SALE-LINES-V (J).
END-PREP-SEND.
    EXIT.
```

6. Miscellaneous Primitives

The following miscellaneous service primitives help the user to issue a query at the terminal, or to obtain all information about the terminal from which the interactive program was activated.

6.1 H_QUERY (QUERY ON TERMINAL)

Purpose:

H_QUERY is used to (re)issue a query at the terminal, until a string meaning "yes" or "no" in the current language is returned. For this purpose it uses the system variables #YES and #NO.

Syntax:

```
$H_QUERY i_charn LENGTH=i_fb15 REPLY=ov_char1;
```

Parameters:

i_charn	Specifies the text of the query.
LENGTH	Specifies the length of the text, which must be greater than 0 and less than 256.
REPLY	Specifies the area which will receive the translated reply: 1 is yes, 0 is no.

Return codes:

Normal:

DONE

Abnormal:

ARGERR	invalid argument (length error)
INDERR	system variable #YES or #NO is without a value

6.2 H_DCTNATTR (DECLARE TERMINAL ATTRIBUTES)

Purpose:

H_DCTNATTR is used to declare the structure filled by the H_TNATTR primitive.

Syntax:

```
$H_DCTNATTR [ PREFIX = l_identifier16 ] [ ATTRIB = l_char ];
```

Parameters:

PREFIX	Specifies a character string which is prefixed to the name of the structure and to the name of each elementary item. The default value is "H_". ATTRIB specifies the attributes of the structure. There are no default attributes.

Comments:

This primitive returns all the information about the terminal from which the interactive program was launched. The information is valid only if IOF="01"X.

The LINE_LENGTH and PAGE_LENGTH parameters reflect the logical lengths as defined by the corresponding system variables. They may be less than or equal to the actual terminal lengths.

Example:

The macro statement $H_DCTNATTR generates this structure:

```
/*------ $H_DCTNATTR; *V*/

DCL 1 H_TNATTR
  2 H_IOF               BIT(8),          /* 00: Batch, 01: IOF   /*
  2 H_MODEL             BIT(16),         /* Model Nb             /*
  2 H_DVTYPE,
   3 H_DISPLAY          BIT(1),          /* CRT Terminal         /*
   3 H_KEYBRD           BIT(1),          /* Keyboard exists      /*
   3 *                  BIT(6),
  2 H_LINE_LENGTH       LOGBIN(8),       /* Terminal line length /*
  2 H_PAGE_LENGTH       LOGBIN(8),       /* Terminal page length /*
  2 H_ENV FLAGS,
   3 *                  BIT(1),
   3 H_FORM_SUPPORT     BIT(1),          /* May be used in       /*
                                         /* formatted mode       /*
   3 *                  BIT(30),
  2 *                   CHAR(12),
  2 H_DVFEATR,
   3 *                  BIT(2),
   3 H_ROLLUP           BIT(1),
   3 H_WRAPAROUND       BIT(1),
   3 *                  BIT(2),
   3 H_AUTOLF           BIT(1),          /* Automatic line feed  /*
                                         /* after input          /*
   3 H_LINFOLD          BIT(1),          /* Automatic line       /*
                                         /* folding              /*
   3 *                  BIT(6),
   3 H_HTAB             BIT(1),          /* Horizontal tabulation/*
                                         /* capability           /*
   3 H VTAB             BIT(1),          /* Vertical tabulation  /*
                                         /* capability           /*
  2 *                   CHAR(8);
```

6.3 H_TNATTR (GET TERMINAL ATTRIBUTES)

Purpose:

H_TNATTR is used to get the execution mode (batch or IOF). In the case of IOF, H_TNATTR returns all information about the terminal from which the interactive program was activated.

Syntax:

```
$H_TNATTR o_structure;
```

Parameters:

o_structure	Specifies the name of the structure in which the information is returned. This structure must be declared by the H_DCTNATTR primitive.

Return codes:

Normal:

DONE

Abnormal:

NONE

Index