Display Input/Output Facility Version 3
Program Description and Operations Manual
Release 1.0

Document Number SB11-8419-0

May 30, 1989

Editor: Sandy Jacobi

First Edition (April 1989)

This edition applies to Release 1, Modification Level 0, of Display Input/Output Facility Version 3, Program Number 5785-HAX, and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, 4300, and 9370 Processors: Bibliography of Industry Systems and Application Programs*, GC20-0370, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to:

IBM International Operations
International Field Program Center
P.O. Box 24
1420 AA Uithoorn
The Netherlands

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

The IBM Display Input/Output Facility Version 3 (5785-HAX), referred to in this manual as DIOF-3, enables users to benefit from the functions of IBM 3270 Display Stations without the need to program them. Potential users are:

- End users writing simple, full-screen EXECs
- Application or system programmers writing full-screen front-ends to application programs or online Help facilities.

This manual describes DIOF-3, explains how to run it, and guides the reader through the installation phase. Please note that this manual refers to the main module of DIOF-3 as IOS3270.

The chapters covered are:

Chapter 1, "Introduction"
Gives a brief presentation of DIOF-3 and explains the purpose of the facility.

Chapter 2, "Understanding DIOF-3"
Describes the input used by and the output returned from the main module of DIOF-3 (IOS3270).

Chapter 3, "Installation"
Provides a step-by-step description of how to install this program.

Chapter 4, "Running the IOS3270 Program"
Describes, in detail, all actions needed to perform specific functions.

Appendix A, "Questions from IOS3270 Users"
Gives the answers to frequently asked questions.

Appendix B, "Summary of Function Characters"
Summarizes all the function characters used with the IOS3270 program.

Appendix C, "The IOSLIB Program"
Describes a program to create and maintain libraries.

Appendix D, "The GOODIES/GDCSS Package"
Describes a package for shared segment operations.

Appendix E, "The IOSDYNAT EXEC"
Shows the IOSDYNAT EXEC, used for dynamic attribute changes.

Appendix F, "The IOSIVP6 EXEC"
Shows the IOSIVP6 EXEC, provided as an example of using the STEM, FROM, and FOR options.

An index is provided at the back of the manual.

Contents

Figures

Summary of Changes for Version 3

The following changes have been made for Version 3:

Support for New Releases of VM

- VM/SP 3 support added
- VM/XA migration aid support added
- VM/SP 4 support added
- VM/XA SF 2 support added
- VM/XA SP 1 support added
- VM/SP 5 support added (CONSOLE MACRO)
- VM/XA SP 2 support added (CMS5.5 support).

New Device Support

- IBM 3290 information panel support.

Performance Related Changes

- Improved performance on VM/SP 3 using DIAG X'8C'.
- IOSLIB now creates variable record length files to reduce the amount of disk space required to store libraries. IOS3270 handles both types of IOSLIBs.

  Old IOSLIBs for which no source is available can be converted using the IOSLIB COMP function.
- Improved performance imbedding members of the same IOSLIB.
- Option added to INSTGOOD to specify whether the DCSS should be kept attached or purged on completion to reduce the (CP) overhead involved in DCSS processing on systems with many users.
- Performance improvement for repetitive .i.

Added Functions, Options, and Attributes

- System Product Interpreter (REXX) support added.
- Field attribute type Nulls now implies Unprotected.
- Minus prefix handling for function character L similar to XEDIT.
- Field attribute type Tab added to allow use of Tab keys.
- QUERY option added to retrieve DIAG X'8C' info.
- The maximum number of input- and selector light pen fields is increased (was 123). The maximum number of combined input-, selector light pen fields and .w/x functions is now (rows*columns/7)-1. Rows and columns refer to the usable area of the screen, not to the size of the displayed panel.
- Maximum IOSLIB dictionary size doubled to 64K.
- BIND option allows for left margin (like SCRIPT).
- Field attribute type Numeric added.
- FBLOCK support added (a general in-storage interface compatible with EXECLOAD).

- Attribute type Alignment introduced to increase the number of fields which can be placed on a line by compressing adjacent attribute bytes (Attributes compress) or dynamic adjustment after variable substitution (Variables float).
- Function character V can now be used to suppress variable substitution.
- Attribute type Scope and .jx Set Mask added to enable the use of character attributes.
- Attribute type Dynamic added to dynamically
  1. Redefine field attributes
  2. Assign character attributes to the individual characters within a field and
  3. Position the cursor under any character within a field.
- The GRAF option added to allow displaying panels on terminals other than the virtual console (ATTACHed or DIALed).
- Attribute type Etmode added to provide DBCS support (limited).
- STEM option and accompanying FROM and FOR options added. (Similar to EXECIO.)

Chapter 1. Introduction

Display Input/Output Facility Version 3 (DIOF-3) is designed to use the functions provided by the family of IBM 3270 Display Stations, without the need to program them.

Potential users are end users writing simple full-screen EXECs and applications or system programmers writing more complicated application programs and online Help facilities.[1]

Input may be static, for example, a Conversational Monitor System (CMS) file, or created dynamically in virtual storage (either directly by, for example, an Assemble program or indirectly in the form of REXX symbols). Most users will be able to find a way to utilize the capabilities of DIOF-3 to suit their needs.

It is important that new users do not attempt to learn all the functions available until the need arises. Becoming aware of the various functions provided and referring to the manual later for specific functions as they are needed is an effective way to familiarize yourself with the program.

DIOF-3 is primarily written to display data entry panels, also referred to as (selection) menus. A panel can be used to give explanations or to ask the operator to specify required parameters to perform a function. Normally, it will be a combination of the two.

With DIOF-3, you can set up a panel and, using the screen, ask questions and provide information at the same time. You can inform operators about input errors by inserting an error message in the panel and positioning the cursor in the field in error. Asking questions and providing information in this way is usually clearer and thus easier to do, and results in fewer input errors than with a line-by-line mechanism.

---

[1] Where the scope of DIOF Version 1 ranged from simple user EXECs to complex system applications, today it seems appropriate to redirect complex (high-end), long-term applications requiring full compliance to IBM SAA to more suitable programs such as IBM's ISPF and/or GDDM/REXX. For Help-type applications (displaying Help panels), the CMS native HELP function would be the correct choice.

Where these programs provide significantly more function (such as dialog management in the case of ISPF), they also require more effort to implement. When applications do not need to fully comply to IBM's SAA, such as small, short-term (user) applications and special applications, users will find it beneficial to use DIOF. The short learning curve will quickly allow users to write small full-screen applications, where adding a full-screen interface would otherwise not have been considered.

Chapter 2. Understanding DIOF-3

The following steps form the basic flow of DIOF-3:

1. Read caller's input
2. Process and format the input according to the caller's specifications
3. Display the result and wait for an interrupt, normally an operator action
4. Process operator input and pass it to the caller according to his/her specifications
5. If there is more than one panel of caller's input, return to step 1.[2] If there is no input left, return to the caller.

2.1 Input to DIOF-3

There are two types of input:

1. Input from a program using the facility.
2. Input from a panel with data entered by a terminal operator.

Input to DIOF-3 from the calling program may be:

- A CMS file
- An array of REXX variables
- A CMS program stack
- A block of virtual storage using one of the two in-storage interfaces.

or a combination of these.

All types of CMS files can be used as input for DIOF-3. However, it never reads more than the first 160 characters of a record. These characters are then formatted and as many as can fit on a screen row (line) are displayed. Because the first position on a row is used by DIOF-3, this will be the screen width minus one character. If the length of a formatted line exceeds the screen width, the output line is truncated. This may lead to incomplete fields, for example, input fields defined with the cursor skip attribute which do not skip. It is the user's responsibility to prevent such an occurrence.

Sometimes it is necessary to read more characters than can fit on a screen row. An example is an input line containing variable names that are longer than their values. The input line may be 100 characters, but after substitution of the variables, there may only be 60 left.

Input to DIOF-3 consists of three types of records:

1. Records that (with or without formatting) that are to be displayed.
2. Records that contain a label.

   A record is considered a label record if it starts with a semicolon (;) in column 1. (For further details refer to Chapter 4, "Running the IOS3270 Program" on page 8.)

[2] Since DIOF-3 is primarily designed as a panel manager and not as a dialog manager, it is not advised to use this mode of operation for data entry panels. For data output panels, there are some primitive scrolling capabilities.

3. Records that contain functional information.

   A record is considered a function record if it starts with a period (.) in column 1. A function record contains function characters, discussed in 4.4, “Function Characters” on page 14.

Records with a semicolon (;) or a period (.) in column 1, are normally not displayed on the screen.

2.2 Output from DIOF-3

There are two types of output:

- Output displayed on a screen from the program
- Output from the screen entered by the operator.

All rows of a supported IBM 3270 Information Display System screen can be used. The row length is either one or two characters less than the screen width, depending on the type of the field that is displayed in the next to the last column in a row.

No default input area is created, and pressing the ENTER key is the signal to continue processing.

Output from DIOF-3 from data entered on panel input fields may be returned as:

- Variables
- User’s virtual storage
- A CMS program stack
- A CMS file.

or a combination of these.

2.3 Error Messages

The IOS3270 program does not generate error messages. Severe error conditions are reflected to the caller as return codes, explained in 4.3, “Return Codes” on page 14. Unknown, invalid or conflicting parameters, options, or function characters are simply ignored and processing continues.

Chapter 3. Installation

This chapter gives a step-by-step description of how to install Display Input/Output Facility Version 3 (DIOF-3).

3.1 Installation Prerequisites

Machine Requirements

This program is designed to operate on an IBM System/370, 30XX, or 438X host processor. Display terminals supported are all terminals compatible with 3270 data stream display. The APL/TEXT character sets are not supported.

Programming Requirements

Display Input/Output Facility Version 3 is written for assembly by Assembler-H Version 2 (5668-962) and is designed to operate with VM/SP CMS (5664-167) Release 4 or 5 or VM/XA SP (5664-308) Release 2 with CMS Release 5.5 (Bimodel CMS). Later versions or releases could affect the functioning of the program. This program, as distributed, does not require compilation.

Note: Since its creation for VM/370 Release 3, IOS3270 has run on all versions of VM/CMS, including VM/XA-MA/SF and VM/PC. Installation on earlier versions of releases is still possible, but may require minor source code modifications. Specifically, the reference of CMS control blocks added in later versions will have to be handled. Only installation on the above-mentioned programs is supported.

3.2 Dependencies

The IOS3270 program has the following dependencies:

1. The data returned from the DIAGNOSE 24 instruction to determine the terminal type and model. This data has changed twice since the introduction of VM/370 Release 5, to introduce new codes for the IBM 3278/79 type screens. The last change was introduced with VM/370 Release 5, PLC 7.
2. The layout of the EXEC interpreter's (DMSEXT) work area. As there is no external interface to this area, the DSECT reflecting it must be corrected every time it changes. Since DIOF-3 was written, it has changed with the introduction of VM/370, Release 3, PLC 20, and with the introduction of BSEPP 2.1 (VM/370, Release 6, PLC 1).
3. The EXECCOMM interface introduced with VM/SP Release 2. It has changed with the introduction of VM/SP Release 3 to add support for REXX symbolic names.
4. The data returned from the DIAGNOSE X'8C' instruction to determine the terminal characteristics and features. Introduced with VM/SP Release 3.

3.3 Machine-readable

DIOF-3 is distributed on a single reel of tape. The files on the tape can be loaded by mounting the tape on an appropriate tape drive, accessing the required disk as A-disk and issuing the TAPE LOAD command. The tape contains two files, separated by a tape mark.

File 1

IOS3270	MODULE	A2
IOS3270	XMOD	A2
IOS3270	IOS3270	A2
IOSPAGE	IOS3270	A2
IOSLIB	MODULE	A2
IOSLIB	IOS3270	A2
IOSDYNAT	EXEC	A2

File 2

IOS3270	ASSEMBLE	A1
IOS3270	AUXIOSPL	A1
IOS3270	AUXIOSLO	A1
IOS3270	TEXT	A1
IOSLIB	ASSEMBLE	A1
IOSLIB	TEXT	A1
GDCSS	ASSEMBLE	A1
GDCSS	TEXT	A1
GOODEND	ASSEMBLE	A1
GOODEND	TEXT	A1
GOODIES	ASSEMBLE	A1
GOODIES	TEXT	A1
INSTGOOD	EXEC	A1
IOSEXEC	EXEC	A1
IOSEXEC2	EXEC	A1
IOSREXX	EXEC	A1
IOSIVP1	EXEC	A1
IOSIVP2	EXEC	A1
IOSIVP3	EXEC	A1
IOSIVP4	EXEC	A1
IOSIVP5	EXEC	A1
IOSIVP6	EXEC	A1
IOS	CNTRL	A1

The files from File 1 should be placed on a disk that is available for those persons authorized to use DIOF-3, for example, the system extensions disk, the Y-disk. The files from File 2 are not needed to run DIOF-3 and may be placed elsewhere for reference and/or maintenance.

Installation is now complete.

Successful installation can be tested by entering **IOS3270 ?**, which displays a short online user's guide.

The file IOS3270 IOS3270 contains this guide and may be used as an example of an input file. This will test the functional parts of DIOF-3 without using the variable interface.

If the test runs error-free, installation has been successfully completed.

If you wish to run IOS3270 from a discontiguous shared segment (DCSS) (called saved segment on VM/XA), refer to Appendix D, "The GOODIES/GDCSS Package" on page 53 for installation. To improve performance, DCSS installation is highly recommended.

Notes:

1. A number of EXEC files are available to further test the functioning of the program. They can also be used as examples of how the various functions can be used. These files were used during the development of the functions which they address and do not necessarily comprise complete working programs. None of them need to be executed to complete the installation process.

IOSEXEC	**EXEC**	Tests the EXEC variable interface.
IOSEXEC2	**EXEC**	Tests the EXECCOMM variable interface with EXEC 2.
IOSREXX	**EXEC**	Tests the EXECCOMM variable interface with REXX.
IOSIVP1	**EXEC**	Demonstrates the possibilities of the dynamic attribute functions.
IOSIVP2	**EXEC**	Demonstrates the possibilities of attribute type Alignment.
IOSIVP3	**EXEC**	Demonstrates the possibilities of attribute type Etmode.
IOSIVP4	**EXEC**	Demonstrates the possibilities of the STEM, FROM and FOR options.
IOSIVP5	**EXEC**	Is a variation of IOSIVP4.
IOSIVP6	**EXEC**	Is a variation of IOSIVP4.

2. The PF key settings for the online user's guide are defined in file IOSPAGE IOS3270. If required, this file can be updated to provide those PF key settings that are generally used on your system.

3.4 Modifying DIOF-3

DIOF-3 was written for assembly by Assembler-H Version 2 (5668-962).

The IOS3270 source is distributed as an Assemble file and optional update files. The update files have a file type of FxxxxHAX, where *xxxx* refers to a unique IOS3270 update number. The update files can be merged with the Assemble file by using the CMS UPDATE command as described below. For further details on the UPDATE command, refer to the *IBM VM/SP CMS Command and Macro Reference Manual*, SC19-6209, or the *IBM VM/XA SP CMS Command Reference Manual*, SC23-0354.

To regenerate the IOS3270 MODULE, the following CMS command sequence can be used:

```
UPDATE IOS3270 ASSEMBLE A1 IOS CNTRL A (CTL
GLOBAL MACLIB DMSSP CMSLIB DMSSP55 OSMACRO
HASM $IOS3270
RENAME $IOS3270 TEXT A IOS3270 = =
INSTGOOD IOS3270
```

The DMSSP55 MACLIB is the (renamed) DMSSP MACLIB as distributed with VM/XA SP Release 2. In order to allow the creation of an execution module that runs on all supported releases, it is necessary to access the VM/XA SP DMSSP MACLIB as indicated (renamed to DMSSP55 MACLIB) and perform compilation on a VM/SP Release 5 system. Compilation cannot be performed without the VM/XA SP macros.

The HASM command is to be substituted by a command (or command sequence), invoking Assembler-H.

The IOSLIB MODULE can be regenerated by issuing the following CMS commands:

```
GLOBAL MACLIB DMSSP CMSLIB DMSSP55 OSMACRO
HASM IOSLIB
LOAD IOSLIB (CLEAR
GENMOD IOSLIB MODULE A2
```

Chapter 4. Running the IOS3270 Program

This chapter describes how to invoke the IOS3270 program and gives a step-by-step explanation of all function characters used to control program operation. At the end of this chapter, some performance considerations are mentioned.

4.1 Invoking The IOS3270 Program

The IOS3270 program is called with the following parameter list:

IOS3270 [*fn***[***ft***[***fm***]]] [***;sl1***[***;el1***[... ***;eln***]]][(***options***)]]**

Where:

fn Is the file name of the file to be displayed.

ft Is the file type of the file to be displayed. The default file type is IOS3270.

fm Is the file mode of the file to be displayed. It defaults to *, which results in the first file found with specified file name and file type in the standard CMS search order.

sl1 Is the first start label where processing must begin.

el1 Is the first end label where processing must stop.

eln Is the *nth* end label.

When processing a CMS file, **start** and **end** labels may be specified. A start label defines the point from which input records have to be processed. An end label defines the point where processing must be stopped.

Labels may be specified in upper- or lowercase, must start with a semicolon (;), and are limited to a length of eight (8) characters, including the semicolon. If a start label is specified, but no end label, the first label found after the start label is treated as if it were the end label.

Up to ten (10) pairs of start and end labels may be specified.

You may use the same name for start and end labels if they occur several times in the file. To find a label, IOS3270 simply scans the input from the current line towards the end of the file. For this reason, a start label that is located in front of the previous end label will not be found.

If no file name is specified when the IOS3270 program is called, it assumes the input is in the CMS program stack.

Note: If the input file is EXECLOADed, IOS3270 will automatically process the file from storage if the file mode is either not specified or specified as “*”. If a file mode is specified, the file will be read from disk.

4.2 Options

Options are divided in two groups, a preferred group and a compatibility group. The latter is not recommended to be used by new users because similar function can now be achieved by using native CMS services. Another category of compatible options was needed to overcome the limited possibilities of EXEC. With the power of REXX, they do not appear to be required anymore.

Preferred Group

ALARM Sounds the audible alarm once on display of the first panel.

BIND n Shifts the output *n* columns to the right. In combination with the UPDATE option, this allows for pop-up windows leaving data displayed in columns 1 through *n*.

FOR numvar Is the number of variables to be processed. To be used in combination with the STEM option. The default is to process the number of variables indicated by the variable stem0.

FROM varno Is the number of the first variable to be processed. To be used in combination with the STEM option. The default is to start processing variable stem1.

GRAF [c]cuu Specifies that device *ccuu* is to be used to display the output. The specified device designates an ATTACHed or DIALed terminal. The default is to use the console. If specified, the address should not be the address of the console.

LIB ln Specifies that the input file name refers to a member in an IOSLIB. Where *ln* is the file name of the IOSLIB. The file type is always IOSLIB in this case and does not have to be specified. The member to be selected is specified by *fn* in the parameter list. An IOSLIB can be built and maintained with the IOSLIB program, discussed in Appendix C.

NOCLEAR Does not clear the screen on entry. If called without the NOCLEAR option, IOS3270 will first clear (erase the contents of) the screen and then write the new panel. If already in full-screen mode (from a previous call to IOS3270 or another full-screen program), there is no need to clear the screen first. Clearing leads to unnecessary and irritating blinking because VM always wants to write its RUNNING status after clearing the screen.

Note: When running in full-screen mode, it is upsetting to be thrown out of full-screen mode when an ATTN key is pressed and CP wants to deliver some messages. Also, once in full-screen mode, the operator does not know there is a message waiting until he/she presses a key. This is not only an IOS3270 problem, but is also true for any full-screen function, for example, XEDIT. Both problems can be prevented by using the CP TERM BREAKIN GUESTCTL command.

NOSCREEN Specifies that IOS3270 should not check the existence (and features) of a physical device. Instead, a default screen size of 24 rows of 80 columns is assumed. All features used in a panel definition are assumed to be present.

To be used in combination with the SCRIOSD option.

NOQUIT Prevents checking for the IOS3270 quit function. See 4.11, “Special Values In Fields” on page 36.

PA1 {EXIT | IGNORE}

Specifies the action taken when the PA1 key is pressed. The default action is no action. CP puts you into CP READ (console function mode). You can now enter CP commands. Use the CP BEGIN command to return to IOS3270.

EXIT Causes IOS3270 to terminate immediately. &IOSK is set to PA1.

IGNORE Resets the SYSTEM (INPUT INHIBITED on 3277) indicator and sounds the audible alarm.

Notes:

1. Because the screen does not return input data when the PA1 key is pressed, all input entered will be lost.
2. From VM/SP Release 5 onward, the CP TERMINAL BRKKEY should not be assigned to PA1 in order for this option to work as specified.

PA2 {EXIT | SUBSET}

Specifies the action taken when the PA2 key is pressed.

The default action is no action. The SYSTEM (INPUT INHIBITED on 3277) indicator is reset and the audible alarm sounds.

EXIT Causes IOS3270 to terminate immediately. &IOSK is set to PA2.

SUBSET Places the virtual machine in CMS SUBSET mode. Use the CMS RETURN command to return to IOS3270.

Note: As the hardware does not return input data on pressing the PA2 key, all input entered will be lost.

SCRIOSD Indicates that no data is to be displayed. Instead, the complete 3270 data stream is placed in the variable &IOSD. The data stream will be based on the features supported by the console or the specified output device (using the GRAF option), unless the NOSCREEN option is also specified.

STEM xxxx Indicates that the variables *xxxxn* are to be used to supply input data. The variables used are a concatenation of the specified stem and a number that corresponds to the number of input lines. The special variable *xxxx*0 must be set to the number of input lines. The FROM and FOR options may be used to explicitly specify the number of the first variable and the total number of variables to be processed. The default is to start with variable *xxxx*1 and process the number of variables indicated by *xxxx*0. The maximum length variable name is 8 characters.

Note: The seemingly restrictive short length of the variable name can be extended by using a symbolic tail, for example, STEM stem.t. , where *t* is defined dynamically during execution. For a

detailed example using this technique, refer to Appendix F, “The IOSIVP6 EXEC” on page 61.

TIME n

Leaves the panel displayed for a maximum of *n* seconds.

Return to the caller is either by operator request, for example pressing ENTER, or when the specified time has elapsed.

If return to the caller is by operator request, the effect is the same as when the TIME option is not specified.

If return to the caller is made when the specified time has elapsed, &IOSK is set to TIME. All data entered by the operator is returned to the caller, but care should be taken not to handle this as valid data. The operator may have been interrupted by the elapsing timer and had no chance to complete his/her input.

Notes:

1. The TIME option performs a CP SET TIMER REAL/ON sequence on VM/SP and VM/XA systems running CMS Release 5 or lower.
2. When the HNDEXT option is used in combination with the TIME option, the specified routine is not given control when a timer interrupt occurs.

UPDATE

Updates an existing full-screen.

Implies option NOCLEAR. If the display is not in full-screen mode, IOS3270 exits with a return code of 142 (X'8E').

Notes:

1. Use the .L function to start overwriting the screen at a specific line.
2. Use the BIND option to start overwriting the screen at a specific column.
3. When using this function, be prepared to re-construct the complete panel when you get a 142 return code. IOS3270 is not aware of any data on the screen other than what it just wrote, including input fields. Data entered in input fields not created by the last write is lost.

VERSION

Returns the following information about the current level of IOS3270 in &IOSD (REXX only):

```
IOS3270 V3.0 R5. ppp. ll mm/dd/yy
```

It lists the version number, the release number, the PLC number (number of the last fix), the local update number (number of the last local update), and the compilation date. It can be used to test whether functions included only at a certain level are available.

Note: Other versions of IOS3270 that do not support the VERSION option will yield return code 1.

rrcc[c]

Is a four- or five-digit decimal number specifying the cursor address, where *rr* is the row number and *ccc* is the column number.

The default cursor position is at the first defined input field or at

the top left position on the screen (row 1 column 1), if there is no input field.

If *rr* is 00, *ccc* specifies that the cursor is to be set at the *ccc*'th input field.

Compatibility Group

HNDEXT Specifies that the user wants to handle external interrupts.

This option can only be used in combination with the in-storage interface. For a detailed description, refer to Section 4.14, "Using The IOS3270 Options HNDEXT and HNDINT" on page 42.

Due to the restructuring of external interrupt handling on Bimodal CMS systems (CMS 5.5), this option may not work under certain conditions. Only those interrupts will be handled for which no specific external interrupt handlers were specified.

See also the TIME option.

HNDINT Specifies that the user wants to handle I/O interrupts other than from the console.

This option can only be used in combination with the in-storage interface. For a detailed description, refer to Section 4.14, "Using The IOS3270 Options HNDEXT and HNDINT" on page 42.

Because the full-screen I/O logic uses the CMS console manager (when running on VM/SP Release 5 or later), IOS3270 no longer handles I/O interrupts. Therefore, the HNDINT option is no longer supported when running on CMS Release 5 or later.

KEEPFILE (fn [ft [fm]])
Specifies a file ID to be used by the keep function.

The default file ID is **$IOS3270 $KEEP$ A1**.

Note: This option does not force the .K function or override the NOKEEP option.

LIFO Stacks input fields LIFO (default is FIFO).

This option only applies to those input fields that are specified to be returned in the CMS program stack.

NOKEEP Does not use saved input fields.

It may seem unnecessary to carefully save input fields and not use them, but sometimes it is not; for example, for users who wish to start a dialog with an empty panel but also wish to re-display the input if an error is made. If input fields are specified as variable names, this is done automatically. If not, this option can be used selectively.

NOWAIT Does not wait for operator action; exits immediately after writing to the screen.

This option may be used to display a panel for a certain period and continue processing without the need for operator intervention.

Note: This option is for compatibility with earlier versions of IOS3270. The TIME option is suggested to accomplish this function when writing new applications.

QUERY Returns device-dependent information concerning the console, using DIAG X'8C'. For use with REXX or EXEC 2, it returns the following string in &IOSD:

```
www hhh [HIG] [COL] [PSS]
```

where:

www Defines the screen width, number of columns in a row.
hhh Defines the screen height, number of rows.
HIG Indicates that extended highlighting is supported.
COL Indicates that color is supported.
PSS Indicates that programmable symbol sets are supported.

The following REXX procedure can be used to interpret the QUERY reply:

```
/* Subroutine or stand-alone - twa */
qry_d8c:
 /* Query the console using DIAG X'8C' in IOS3270.
    On return, the following variables have been set:
    d8c_width  - www  screen width (#of columns)
    d8c_height - hhh  screen height (#of rows)
    d8c_highl  - 1(0) extended highlighting (not)
                      supported
    d8c_color  - 1(0) color (not) supported
    d8c_symbl  - 1(0) PSS (not) supported
  */
 Address Command 'IOS3270 ( QUERY'
 Parse Var iosd d8c_width d8c_height .
 d8c_color=Sign(Find(iosd,'COL'))
 d8c_highl=Sign(Find(iosd,'HIG'))
 d8c_symbl=Sign(Find(iosd,'PSS'))
 Parse Source . how .
 If how='COMMAND' Then Say iosd
 Return
```

Notes:

1. This option is for compatibility with earlier versions of IOS3270. Using the REXX function Diag(8c) is suggested to accomplish this function when writing new applications, as follows:

   ```
   /* Set major console characteristics */
    d8c=Diag(8c)
    scr_cols =C2d(Substr(d8c,3,2))
    scr_rows =C2d(Substr(d8c,5,2))
    scr_color=Bitand('80'x,Substr(d8c,1,1))='80'x
    scr_highl=Bitand('40'x,Substr(d8c,1,1))='40'x
    scr_pss  =Bitand('20'x,Substr(d8c,1,1))='20'x
   ```

2. As REXX does not allow a device address to be specified, the QUERY function is still useful for ATTACHed or DIALed devices.

4.3 Return Codes

1 Specified input not found.

2 Unsupported console type or disconnected.

3 Display I/O error. The usual cause of this type of error is invalid data in panel variables, in many cases originating from a (corrupted) GLOBALV file. All code points from X'00' through X'3F' and X'FF' are 3270 control characters and should be avoided as data.

4 "Quit" found in an input, selector light pen or PF key field

5 Insufficient virtual storage.

50 Input stacked. (Other return codes may have been added).

142 The screen was not in full-screen mode while trying to modify it, using the UPDATE option.

1xxx xxx is the return code from FSREAD reading input file. (Other return codes may have been added).

4.4 Function Characters

You may insert function characters in the input stream to control the processing of IOS3270 in a similar fashion to Document Composition Facility (DCF) control words. To indicate that a line contains these characters, it should start with a period (.) in column 1. The first blank on a function line (which is not part of a function) ends the scan for other function characters on that line.

Function characters may be any combination of upper- and lowercase alphabetics separated by a semicolon (;). After the first function character, semicolons and periods may be used in any combination or left out. The following are all valid notations:

```
.p;.s  .p.s  .p;s  .ps
```

Function characters with an undefined number of parameters, (for example, .D, .F, .I, .J and .V) must be specified as the last function character on a line.

In most cases, parameters for function characters can be specified as variables. However, substitution of these variables is restricted to one token only. If, for example, the variable &FILE has the value TEST FILE A1, only the word TEST is substituted. Exceptions to this are the functions .V and .&vname.

Like options, function characters are divided in two groups, a preferred group and a compatibility group. The latter is not recommended to be used when writing new applications. The primary use of the functions in the compatibility group was to overcome the limitations of EXEC. With the power of REXX, they do not appear to be required anymore.

Preferred Group

Alarm sounding

.A

The audible alarm sounds whenever this panel is displayed.

This differs from the ALARM option in that it sounds the alarm every time the panel is rewritten, whereas the ALARM option only sounds the alarm on the first write.

Bottom title

.B

The next input line is used as a bottom title and is displayed on the last line of this and every following panel.

A bottom title is automatically displayed intensified (brighter), unless .C is active and .H not. Although you can change a bottom title by inserting another .B, there is no way to completely reset it.

To insure that a line is displayed on the last row of the screen, you may also use .L-1.

Ctlchar activation

.C

Indicates the start of data to be scanned for field definition or control characters (Ctlchars). A Ctlchar marks the start of a field.

These characters and their functions are:

$ Input field normal intensity
¬ Input field intensified
Input field intensified with automatic skip
¢ Input field non-displayed
@ Selector light pen field
! Selector light pen field intensified
% Intensified field
& Variable name.

The characters shown above are the default characters used. They can be reset or redefined with the .JX and .J functions. In addition, the .JX function allows you to define field definition characters with attributes not provided by the default characters.

A field is delimited by an end delimiter. The default end delimiter is a blank ( ). The .E function may be used to change the ending delimiter.

A dash (-) in a field is translated to a null (X'00'). This allows you to chain fields and allows the 3270 insert mode function to be used in input fields. Also, see the description of the .E function below.

The next function record (a line starting with a period) resets the .C function. It doesn't matter whether anything else is on the line after the period or not.

Note: Substitution of variables in function records is carried out regardless of a .C setting.

Ending delimiter

.E*x*

Defines ending delimiter for fields.

x is the character defining the end of a field. If the delimiter is not a blank ( ), dashes (-) are not translated to nulls. Using this function allows you to have blanks and dashes in a field, which is particularly useful in input fields.

If an input field is in the user's virtual storage (using the in-storage interface to IOS3270), if the input is to be returned to the user's virtual storage, and if the ending delimiter is not a blank ( ), then leading blanks are not deleted. No check is made for any special characters or strings, like > or quit. In addition, the special character " is not recognized. This effectively means that input fields are returned to the caller exactly as they are typed in by the operator.

To make sure that the operator cannot influence the length of an input field, a hexadecimal value that cannot be entered by the operator should be used as the delimiter character. A useful value normally is in the range X'FA' to X'FE'.

For example, the input stream:

```
.ce!
Address:%===>!¬Toystreet, 3461 GP Joytown, Funnystate -!
```

will generate the following output line:

```
Address: ===>  Toystreet, 3461 GP Joytown, Funnystate -
```

Note that there are two fields which are both ended by the specified ending delimiter (!).

Function key setting

.F[2] [*func1*[*func2* [...[*funcn*]]]]

Sets program function (PF) keys.

The .F function must be the last one on a function line.

func1, *func2*, ... *funcn* are assigned to PF keys 1-n, respectively. Functions are limited to eight characters. All function fields may be specified as variable names. All PF key functions must first be cleared, then set. The first field is set into PF key 1, the second field is set into PF key 2, and so on.

To leave a particular PF key unassigned, use the percent (%) character. For example, the input stream:

```
.f funca % % funcb
```

would only set PF key 1 and PF key 4 and clear all other PF keys.

To set PF keys 13 to 24, specify .F2. The two groups of PF keys (1-12 and 13-24) are cleared and set independently.

If IOS3270 is called from an EXEC, PF key usage is returned as follows: &IOSK is set to the associated PF key used and &IOSD contains the string as specified with the .F function.

If IOS3270 is not called from an EXEC, the string is put in the program stack as a separate line. An exception to this is described in 4.12, “Invoking IOS3270 with In-storage Data” on page 36.

If the .Y function is used, input fields are also processed.

Note: If none of the PF keys 13 to 24 are set, the functions of PF keys 1 to 12 are automatically assigned to them. In this case, &IOSK is set to PF01 when PF key 13 is used. So, if PF keys 13 to 24 are to be handled exactly like PF keys 1 to 12, you just code as if you only had to handle PF keys 1 to 12. Mapping of PF 13 to PF 1, and so on, is automatic and transparent.

High-intensity display of lines

.H[*n*]

Displays next line(s) intensified.

n specifies the number of lines to be displayed (with a higher intensity, (brighter)). If not specified, only one line is displayed intensified. Substitution of fields is done depending on the .C setting.

On color screens, intensified lines (or fields) are displayed in white, or with the attributes (color, highlighting, and programmable symbol set) specified in the last .JX Set High-intensity function. This allows you to define panels that look good on both color and monochrome screens. For example, the input lines below display the error message (&msg) in red on a color screen and intensified on a monochrome screen.

```
.jx Set Norm col=yel, High col=red PS=Bold
Please complete this questionnaire
.ch
&msg
```

In addition, on screens equipped with programmable symbol sets, the character set with an LCID of B is used, if loaded. Data with normal intensity is displayed in yellow on a color screen.

Imbed

.I [*IOS3270 parameter list*]

Imbeds input data.

The imbed function must be the last one on a function line.

Parameters and options may be specified as variable names. Recursive imbed is supported.

A typical use of this function is to imbed PF key functions and titles that are to be the same on many panels. If they must be changed, this need only be done once, in the

imbedded file. An example can be seen in the file IOS3270 IOS3270, which contains a short online user's guide, displayed when IOS3270 ? is entered. The imbedded file, IOSPAGE IOS3270, contains the PF key settings and the bottom title.

Note: If the data to be imbedded cannot be found, IOS3270 just continues processing. This allows relevant information to be displayed, depending on the data available to a particular operator. The same can be accomplished using the .Q and .R functions, but with .I it is not necessary to keep track of the number of lines. The .I function also allows for function records, which are not processed with the .Q and .R functions.

Extended define

```
.JX Set Normal-intensity option1
      High-intensity   option1
      Ctlchar c        option1 option2
      Mask             option3

      Option1:
        OFF
        Alignment=
        Color=
        Etmode=
        Highlight=
        PSid=
        ,

      Option2:
        Dynamic
        Scope=
        Type=
        ,

      Option3:
        n
        Reuse
        ,
```

Extended define.

The .JX function must be the last one on a function line.

The .JX function is designed to use the extended attribute support provided by most newer model display stations. It is also used as a replacement for the .J function to define field definition characters (Ctlchars) with attribute combinations not provided by the .J function.

Operands

Set Normal-intensity
Defines the default attribute values to be used for data displayed with normal intensity.

Only those attribute types defined will be set. Other previously defined types remain in effect. Invalid attribute values cause the affected attribute type to be set to the default value.

Example:

```
.jx Set Norm col=red highl=rev
```

This will display data in reverse video red.

Set High-intensity

Defines the default attribute values to be used for data displayed with high intensity (lines with .H and fields with type=high).

Only attribute types that are defined will be set. Other previously defined types remain in effect. Invalid attribute values cause the affected attribute type to be set to “not specified”.

Attribute types that are not specified are inherited from the last .JX Set Normal-intensity.

Example:

```
.jx Set Norm col=yel highl=rev, High col=red PS=Bold
```

This will display data with a high-intensity attribute in reverse red, using a character set with an ID of B, if loaded. Note that the highlight value is taken from the Set Norm.

Set Ctlchar Defines a field definition character. When displayed, the field following this character has the attributes specified. Attribute types that are not specified are inherited from the last Set Norm function or from the last Set High function if the field has attribute typ=high.

Note that there is a difference here in not specifying an attribute type or specifying a value of Default. Invalid attribute values cause the affected attribute type to be set to “not specified”.

Example:

```
.jx Set Norm col=yel, Ctl $ hig=und typ=(unp skip)
.jx Set Ctl @ col=gre typ=unp
Please enter input%==> $fld1 %==> @fld2
```

This displays the line **Please ...** in yellow. The characters ==> are displayed in white. The word **fld1** becomes an input field with a highlight value of **underscore**, and the color is yellow. In addition, the cursor automatically jumps to the next input field when the operator enters a character in the last position of the field. The word **fld2** becomes an input field. The color is green.

Set Mask Specifies that the following input line is to be treated as a mask, changing the attributes of the characters in the line following the mask that correspond to the position of the mask characters. A mask consists of Ctlchars which define the attributes that need to be applied.

Example:

```
.jx Set Norm col=whi, Ctl @ col=gre
.jx Set Mask
@           @                       @
1 Help      3 End                   7 Next
```

This displays the line **1 Help ...** in white, except for the numbers, which are displayed in green. For a detailed description, refer to 4.13, “Specifying Character Attributes and Dynamic Attributes” on page 40.

Options

To aid reading, an equal sign (=) may be used to connect attribute type and value, for example, col=red or typ=(unp inv ski).

Multiple Set functions may be separated by a comma (,), for example:

```
.jx Set Norm col=pin, Ctl # col=yel
```

OFF Resets all extended attributes to their default values, or resets a Ctlchar.

Alignment Defines output alignment, relative to the input data.

(May be omitted. Can be used to pair with a closing parenthesis. As multiple Alignment keywords may be specified, a closing parenthesis is required when followed by other JX Set functions.

Attributes Defines output alignment for attribute bytes.

Align Explicitly aligns attribute bytes with input data.

Compress Compresses attribute bytes when they are preceded by another attribute byte or a blank (X'40'). This allows you to separate two fields by only one character instead of at least two.

Off Specifies that alignment is to be with input data (if defined with Set Norm or Set High) or is to be inherited from Set Norm or Set High (if defined with Set Ctl). This is the default setting.

, May be used to separate multiple Alignment keywords.

Variables Defines output alignment for variables with a length denominator, as used in input fields.

Align Explicitly aligns with input data.

Float Specifies that an input field will occupy the number of characters specified by the length denominator. The default number of characters is equal to the length of the variable name + 1.

Off Specifies that alignment is to be with input data (if defined with Set Norm or Set High) or is to be inherited from Set Norm or Set High (if defined with Set Ctl). This is the default setting.

) Marks the end of Alignment specifications. May be omitted if there is no data is following on this line.

Color Defines the color value. Valid values are:

Blue
Default
Green
Pink
Red

Turquoise
White
Yellow.

See also 4.15, “Color Aspects” on page 45.

Dynamic Specifies that field attributes, character attributes and a cursor position may be specified dynamically. For a detailed description, refer to 4.13, “Specifying Character Attributes and Dynamic Attributes” on page 40.

Etmode Specifies whether the operator is allowed to create DBCS subfields in an input field (SO/SI creation).

Yes Enables SO/SI creation by operator. Used to selectively enable fields.

No Disables SO/SI creation by operator. Used to selectively disable fields which are globally enabled with .jx Set Norm or Set High. This is the default setting.

Notes:

1. DBCS validity checking is not performed by IOS3270. It is the user’s responsibility to ensure that DBCS subfields are properly enclosed in SO/SI. (A DBCS subfield is a DBCS field (string) enclosed by SO/SI.) DBCS fields requiring truncation are not validated and may give unpredictable results. DBCS subfields requiring truncation are truncated at the SO.
2. Output containing DBCS subfields is handled automatically.

Highlight Defines the highlight value. Valid values are:

Blink
Default
Reverse-video
Underscore.

PSid Defines the Local Character set ID (LCID) of the symbol set to be used.

This may be either a programmable symbol set (RWS) or a read-only (ROS) character set, for example, PSid=1 for APL/Text. The ID can be specified as a single character, for example, **B**, or as a three-digit decimal number, for example, **194**. If specified as a single character, the value will be ORed with X'40' (for alphabetic characters this is the uppercase value of a character). This means that **b** refers to the same set as **B**.

If a character set with the specified ID is not currently loaded, the default character set is used.

Notes:

1. A symbol set ID can be attached to a programmable symbol set when it is loaded. Loading of programmable symbol sets is not supported by IOS3270. The Graphical Data Display Manager (GDDM) function PSLSS(C) may be used to do this.
2. Since an ID can only be either a single character or a three-digit numeric value, you are allowed to extend the definition somewhat to make it more readable. Suppose you have loaded a character set BOLD with an LCID of 194 and a set SCRIPT with an LCID of 226. You can refer to these sets by PSid=194 and PSid=226, or PSid=B and PSid=S. However,

this is not very readable. Using a notation PS=194-Bold, PS=B-Bold or simply PS=Bold (PS=226-Script, PS=S-Script, PS=Script) makes it a lot easier to quickly see what is actually wanted. Note that this example only works because B=194 (S=226).

Scope Defines the scope of the attributes to be applied.

Character Specifies that character attributes should be used.

Field Specifies that field attributes should be used. This is the default setting.

Type Defines field attributes.

(

May be omitted. Can be used to pair with a closing parenthesis.

Detectable

Defines the field as a selector light pen field. (See 4.5, “Defining Selector Light Pen Fields” on page 32.) To ease the use of the Cursor-Select key, define Tab as well so the operator can make use of the Tab keys.

Escape

Defines an escape character. The character following the escape character is unconditionally displayed on the screen. Use an escape character if you want to display a Ctlchar. You can also use the escape character to display a period (.) or semicolon (;) in column 1.

High-intensity

Displays the field with a higher intensity (brighter). If extended highlighting is also specified and supported by the display, the high intensity value is ignored.

Note: Intensity and highlighting are easily confused:

Intensity is supported only on monochrome, but emulated in white on color screens. Binary value: high|low. In IOS3270, it is specified with **H** for lines and with **.JX Set Ctl ... typ=high** for fields.

Highlighting is supported on most newer type display stations, color or monochrome. Values: underscore, blink, reverse-video. In IOS3270, it is specified with **.JX Set ... High=***value* for both lines and fields. If specified and supported by the display, intensity is ignored.

Invisible

Data in the field is not displayed. This value should be used for input fields intended for password entry.

Nulls

Specifies that input fields are to be left padded with nulls (X'00'). This allows the use of the 3270 INSERT MODE function.

Nulls implies (forces) Unprotected.

NUMeric

Specifies that input fields should have the Numeric Lock Feature activated, when installed.

NUmeric implies (forces) Unprotected.

Skip

When a character is entered in the last position of the field, the cursor automatically jumps to the first position of the next input field. To be used in combination with Unprotected.

Tab

Defines the field as an unprotected field to allow the use of the Tab keys by the operator. This attribute is useful in combination with Detectable to allow easy use of the Cursor Select key. It can also be used in combination with other attributes or on its own for programs using the cursor position as a selection mechanism.

Note that although these fields are unprotected and thus can be modified by the operator, they are never returned to the calling program as being operator input fields.

Unprotected

Defines the field as an input field. For input fields which are used to enter passwords, the attribute type Invisible should be specified as well.

)

Marks the end of Type attribute values. May be omitted if there is no data following on the line.

n Defines the number of input lines to which the mask should be applied. The default is one.

Reuse Specifies that a previously defined mask should be used. The next input line is not treated as a mask in this case.

Notes:

1. The .j and the .jx Set Ctlchar functions can be used in combination. Use of either form resets previously defined attributes for the same character.
2. For input fields, the FILL character is omitted when:
   - A FILL character '_' has been specified
   - Extended highlighting is supported
   - The field has been specified with a highlight value of underscore.

   This reduces the amount of data sent to the screen and produces a clearer indication on terminals supporting extended highlighting.

   Automatic insertion of a FILL character _ for devices not supporting extended highlighting, while underscore was requested, is deliberately not done to reduce the amount of data sent to the screen.
3. Currently, only the first character of each token is inspected (except for NUMeric). Thus, each of the following has the same effect:

   ```
   .jx Set Norm col=red high=rev PSid=i, Ctl * OFF
   ```

   or

```
.jx S nor c=r h=r PS=i, Ctl * o
```

or

```
.jx s n c r h r p i , c * o
```

Besides the fact that the last notation is unreadable, it may be necessary to compare more than one character in future versions. Therefore, you are advised not to invent your own abbreviations. Using a minimum of three characters is strongly recommended.

4. Depending on the VM Release it is running on, IOS3270 uses the following techniques to find out about the features supported by the device.

   VM/SP Release 3 and upward and all Releases of VM/XA SP: A DIAGNOSE X'8C' is issued during initialization, providing information about the availability of extended highlighting, color, and programmable symbol sets (PSS). When use of programmable symbol sets is requested and PSS is available, a write structured fields (WSF) query (using DIAG X'58' or CONSOLE) is issued to retrieve the symbol set IDs (PSIDs) of the loaded programmable symbol sets (if any).

   DIAG X'58' requires the console to be in full-screen mode before it allows the WSF. This normally means the screen has to be cleared and an Erase Write has to be issued before data containing extended attributes can be written.

   Other Releases of VM/SP o r VM/370: Since DIAG X'8C' is not available before VM/SP Release 3, IOS3270 has to get all the information itself, using the WSF Query function. If the device is a 3277, no attempt is made to query, as these devices don't support extended attributes. If the device is not a 3277, the WSF Query is issued. If the control unit does not support this function, this will result in a PROG471 on the affected device. In VM/SP 1.1, DIAG X'58' checks RDEVADVF to determine whether the device has either extended highlight, color or PSS support. If not, it rejects the DIAG X'58' WSF with a UC and COMMAND REJECT.

Line (row) definition

.L*[-]n*

Skips to a specific row (line).

n is a one- or two-digit row number where writing is to continue. If prefixed by a minus sign (-), the next line is displayed *n* lines from the bottom of the screen. Be aware that specifying **.l-1** allows only one additional text line to be read. After that, the screen is considered "full" and IOS3270 displays all data up to that point only.

This function allows for a certain amount of space to be reserved on the panel for optionally imbedded parts that occupy a different number of rows. For example, the following call and file could be used:

```
IOS3270 X ;TOP ;TOPEND &L1 &L2 ;END
;TOP
This is the first line displayed.
.ssss
;TOPEND
;MSG1
This is a message
;MSGEND
;END
.l15
This is the last line.
```

If &L1 and &L2 are not set, this call results in displaying the top and bottom parts of the file. If &L1 and &L2 are set to ;MSG1 and ;MSGEND respectively, this inserts the message part in the right place. The last line is always on row 15.

Maximum screen size use

.M

This function instructs IOS3270 to use the maximum number of rows and columns that are available on a particular model display station. The default panel (page) size is 24 rows of 80 columns for all models.

Note: The .M function takes effect from the point where it is processed. IOS3270 uses this information to calculate relative screen coordinates. Therefore, the .M function should be specified as one of the first functions to prevent misalignment on display stations which have a different number of columns.

No edit

.N

Skips editing data from input fields which are defined as variable names. Normally, leading and trailing blanks are removed and uppercase translation is done. Note that EXEC variable values cannot start with a blank and can only contain one token. This function is primarily meant for programs using the EXECCOMM interface introduced with CMS for VM/SP Release 2.

Note: The .N function also affects variable values set by the .V function.

Page eject

.P

Skips to next panel.

Displays everything processed up to this function character. If there are more function characters after the .P, these will be processed after the operator has caused IOS3270 to continue processing. This is only done if a page eject is explicitly specified by .P. If a page eject is implicitly done while processing a function line, the part of the line following the page eject will not be processed.

Top titles (defined with .T) and a bottom title (defined with .B) are used to format the screen following an implicit or explicit page eject.

Note: Not recommended for panels containing input fields.

Qualified Read/Reserve

.Q

This function is equivalent to the .R function below, but it also allows input and selector light pen fields to be generated.

The reason for this extra function is that additional input from the screen is likely to influence the logic of the calling program. If this is not wanted, it can be prevented by using the .R function instead.

Read/Reserve panel space

.R

Includes data from the program stack or from a file other than the input file without losing control over the panel layout.

This means you can reserve some space on a panel where the operator may include his/her own comments by putting them in a file on one of his/her own disks.

Some space for memos can also be reserved as a kind of log message or “hot news”. (Use the .D function to specify a file to be used.)

Under control of .C, the input is checked for fields other than input and selector light pen. You can allow for input and selector light pen fields by using the .Q function. For every .R, one record is read. If there are no more records, a blank line is inserted for every .R.

Note that function records (.xxxx) and labels (;xxxx) are treated as data when read by the .R or .Q functions.

Space

.S

Spaces a line; displays a blank line.

Note: When the UPDATE option is specified, space is treated as a skip, thus leaving displayed data untouched.

Top title

.T

Specifies ending or reset of top titles.

This function is only useful for multi-panel output applications.

It is possible to have as many top titles as required. Top-title processing is activated when a .T is found on a function line. All lines processed for a page up to the .T will be used as top titles. Whenever a .T is found, a check is made to see if any line was formatted after a page eject. If not, current top titles are reset. The way to reset top titles is to specify .PT. To add lines to existing top titles, first force a page eject (by .P), insert the lines that should be added to the top title(s), and insert a .T.

Variable setting

.V &n1 v1 [&n2 v2 ... [&nn vn]]
Off

Stores variable values.

The .V function must be the last one on a function line.

v1 is assigned to &*n1*, *v2* to &*n2*, and so on. Values may be specified as variable names. Values not specified as variable names are translated to uppercase. If a lowercase value is needed, specify the value as the name of a variable containing the lowercase value, or use the .N function.

Specifying .V Off suppresses variable substitution in the remainder of the input data. Note that this should be used with care as it also prevents the setting of the &IOS*x* group of variables.

.Y

Processes input fields after handling a PF key interrupt.

Normally only the function defined for a PF key is processed. Inserting a .Y instructs IOS3270 to process input fields as well.

.&vname

Substitutes this line with the value of variable *vname*.

The input line is replaced by the value of specified variable and treated as if it were the original input line. If the variable does not exist or has a null value, the input line is simply ignored. This allows you to make dynamic changes to the input file. The variable value can be data, an IOS3270 label, an IOS3270 function record, or even another .& statement. (IOS3270 goes into a loop if the value of a variable is set to its own name, preceded by a period.)

With some functions it is possible to have parameters specified as variable names. This function is practically equivalent to having that possibility with every function.

With the introduction of EXEC 2, it is now possible to have more than eight characters assigned to a variable value, including blanks. This allows you to put a complete function record in a variable.

Note: This function is basically superseded by the STEM option, which provides even greater flexibility and has the advantage that a (dummy) input file need not be specified.

Compatibility Group

Define I/O

.D fn [ft [fm]] [(recno [])]

Specifies an input or output file to be used with the .Q, .R or .W function, where:

fn Is the file name. If *, it resets a previously specified .D function.

ft Is the file type. Defaults to IOS3270.

fm Is the file mode. Defaults to * if used with .R and to A1 if used with .W.

recno Is the record number where reading or writing has to start. Defaults to record 1 for read and to the next available record for write.

All parameters may be specified as variable names.

The .D function may start anywhere on a function line but must be the last one on that line.

If both input and output files are to be specified, insert the definition for output after the last .R or .Q, using the input definition.

Several files may be specified to be used as input files with .R, but only the last one is used for the .W function.

Always specify a file ID as completely as possible, especially if it is used to pick up selective records from a file by resetting the starting record number. If the complete file ID is not specified, this can lead to severe performance degradation due to closing and opening of the same file.

.J [.....]

Redefines field definition characters (Ctlchars).

The .J function must be the last one on a function line.

Field definition characters are used to define the start of fields, as explained in the description of the .C function. The .J function allows you to reset or redefine these characters.

The string of characters following the first blank after the .J is positional. Every position represents a field type. If a position is a blank (X'40') or not specified, the Ctlchar remains unchanged. The ampersand (&) may be used to reset a Ctlchar.

The .J function is processed as follows. The character string is translated to uppercase (ORed with X'40'). Then the character string is scanned, and every character is set up to define a field with the following attributes:

1	(%)	Intensify
2	($)	Input
3	(¬)	Input, intensified
4	(#)	Input, intensified, automatic skip
5	(¢)	Input, non-displayed
6	(@)	Selector light pen
7	(!)	Selector light pen, intensified
8	(.)	Place holder in variable names
9	( )	Fill character for input fields
10	( )	Input, automatic skip

The numbers refer to the relative position in the string; the character between parentheses is the default Ctlchar used to define the field.

Examples:

- If the Ctlchar to identify an intensified field must be changed to **)**, the statement .j) will do that. All other Ctlchars remain unchanged.
- If **:** is to be used to define input fields, and no other Ctlchars are wanted, the statement .j &:&&&&&& will accomplish this.
- Position nine (9) can be used to define a FILL character. This character will replace trailing blank characters in input fields. It is not returned to the caller.

```
Input                : .j         _|
                     : #---- |---- |aa---- ¬a_b----

Displayed on screen :  ____  ____  aa____  a_b____

Modified by operator:  ____  x___  __a___  a_b c__

Returned to caller  : x __a a_b c
```

  See the note about FILL characters in the description of the .JX function.

Keep input

.K[(&index)]

Saves (keeps) input field values and displays them the next time this panel is displayed.

This function saves data entered in input fields. The data is saved in the file, $IOS3270 $KEEP$. The file mode number is 1. The file mode letter may differ. The KEEPFILE option may be used to specify a different file ID. IOS3270 searches all disks in the standard CMS search order to find a keep file. The first one found will be scanned for a matching entry. When IOS3270 wants to write the input fields back to the keep file, it first tries to write it on the disk where the file was found. If that disk is accessed in read-only (R/O) mode, and this disk is not the A-disk, an attempt is made to create a new file on the A-disk.

Kept input on the bottom two lines may be erased with the .Z function. (See also the NOKEEP option.)

A keep entry has four fields to identify a panel:

1. File name
2. File type

3. If the file type is IOSLIB, the member name
4. A user index (&index).

Fields 1 to 3 are the identifiers of the primary input file. Field 4, the user index, is optional and can be used if the same file is used by different EXECs to identify which keep entry should be used. In this example, **&index** could be specified as **&0** to relate the keep entry directly to the calling EXEC's name.

Notes:

1. The Keep function originates from the time when REXX was not available. Input fields longer than eight characters and input fields with imbedded blanks were difficult to process. Keep, at that time, provided a reasonably general way to remember things. Today it seems more effective to have the function handled by the calling program, using GLOBALV. GLOBALV will usually be faster, automatically handles file maintenance, and allows you to use the same input data on different panels.
2. The Keep function cannot be used if the input results in a display of more than one (1) panel.
3. If the layout of a panel is changed, input fields may occur where they are least expected to. In this case, press the ERASE INPUT and ENTER keys to update the keep file to the new panel layout.
4. Keep should only be used when it is needed. As keep data is appended to the keep file, the size of this file increases when new functions are added. This may lead to unexpected performance degradation. To avoid this, the KEEPFILE option may be used to spread the data over different files. From time to time, you may also want to erase large keep files which contain data from functions no longer used.

Omission character

.O *x*

Specifies a character (*x*) to be inserted in the program stack or assigned to a variable for input fields in which no data is entered by the operator. The function character and the specified omission character should be separated by one (1) blank.

This function is used to make it easier to check input fields returned in the program stack. For example, there are four input fields:

```
1=#-----  2=#---- 3=#----- 4=¬----
```

If the operator inserts values A and B in fields 2 and 4, the following appears in the program stack:

```
A B
```

This is fine if it is of no importance in which field data is entered, but useless information if a relation between the input field and the data entered is needed.

Now assume that .O * function is inserted. The program stack will now contain:

```
* A * B
```

giving the relation between entered data and input fields.

There is no indication for trailing empty fields if they should be stacked. If this is required, the special input character " can be used. This effectively results in a forced stack of all input fields prior to this field including the field itself. Variables are always set to either the omission character or blanks for unused input fields.

Note: As discussed with the keep function, this function also originates from pre-REXX time. Today, using REXX variables seems to provide an easier method.

Write panel input

.W

Writes all input from the panel up to this line to a file defined with .D.

The file must be fixed, LRECL 80. If a starting record number was specified with .D, the file is (over)written starting at that record number. If a starting record number was not specified, the file will be created (if it does not exist) or appended to (if it does.) Insert a .W for every record to be written. See also the .X function.

Notes:

1. As IOS3270 stops reading input as soon as the panel is full, it is not possible to use the .W function for input fields defined in the last line on the screen.
2. As discussed with the Keep function, this function also originates from pre-REXX time. Today, using REXX variables and EXECIO seems to provide a more flexible method.

eXtend input buffer

.X

Processes all input fields up to here.

When processing input data from the panel, IOS3270 moves all input data that must go into the program stack (or must be written into a file) into a 132 byte buffer. If this buffer is full, it is put in the program stack and IOS3270 continues processing. Input that could not be put in the buffer is lost.

If you have many input fields on a panel, 132 bytes may be not enough to contain all the entered data. Inserting .X forces IOS3270 to stack all data processed so far and start with an empty buffer again. There will be one line in the program stack for every .X.

Note: As discussed with the Keep function, this function also originates from pre-REXX time. Today, using REXX variables seems to provide a more flexible method.

Zero out (clear) input fields

.Z

Clears input fields on bottom two lines.

If the Keep function is used or an input field is defined as a variable, the input fields entered previously are made available. If an input field defines a special function to be

performed, this field can be omitted to relieve the operator from erasing it every time. This can be done by using the .Z function and placing this field on one of the bottom two lines.

4.5 Defining Selector Light Pen Fields

The first character of a selector light pen field is the designator character. Designator characters are used to define two types of selector light pen fields: selection fields and attention fields. Each type of field performs a different selector light pen operation. Both types of selector light pen fields may be specified as variable names.

Note: Selector light pen fields can also be detected by the Cursor-Select key, which is a basic feature on newer model display stations.

The selection field is defined by a question mark (?) designator character, for example **!?select**. When the selector light pen detects on a selection field, the designator character is automatically changed to a greater-than (>) sign to provide a visible indication to the operator that the detection was successful. If a mistake was made and the operator again detects on that same field, the > changes to a ? and thus resets the erroneous selection. Selected fields are handled as input fields and are returned through the CMS program stack. The fields are stacked left to right, top to bottom.

The attention field is defined by a space ( ) designator character, for example **! attn**. A detection on an attention field normally results in a return to the caller. The value of the detected field is put in &IOSD if called from an EXEC, or in the program stack if not called from an EXEC. Input entered by the operator will not be processed. See the next section if this is required.

A second type of attention field is defined by the ampersand (&) designator character. A selector light pen detection on a field containing an ampersand designator character has exactly the same result as pressing the ENTER key. Note that it is not possible to differentiate between multiple attention fields. Only selected fields and keyboard input is returned to the caller. Selected fields are handled as input fields and are returned through the CMS program stack.

The 3270 hardware puts restrictions on the number of selector light pen fields with an attention designator character that may be defined on one screen. For example, a maximum of 12 detectable fields may precede the last detectable field on any given line. When mixing detectable and non-detectable fields, a maximum of 14 mixed fields may precede the last detectable field on any given line.

The 3270 hardware requires a selector light pen field to have three blank or null characters before and after the field to guarantee good selection. Fields longer than three characters (or fields intended to be used for cursor select only) can do without these requirements. Short fields, for panel selection, for example, must use the IOS3270 null character (-) to make selection possible, for example, **! -1-**.

4.6 Defining Input Fields

A dash (-) in an input field is replaced by a null (X'00').

Trailing omitted input fields are normally not reflected to the caller.

If you do not want the field to be displayed on the screen, use the non-display Ctlchar (¢).

If a check on intermediate empty input fields is required, use the .O function. In conjunction with the .O function you may want to use the special character ". If this character is the first character of an input field, it becomes a protected field. The field is always returned to the caller. To effectively use it in combination with the .O function, it should be defined after the last input field which is to be affected by the .O function.

Input fields are returned to the caller as:

- A variable value
- Values in virtual storage
- CMS program stack
- A file (use .W and .D)

(See also 4.8, “Using Variables in Input Fields” on page 34, and the .E and .J functions in 4.4, “Function Characters” on page 14.

4.7 Using Variables

Variable names (often called symbols in REXX) must start with an ampersand (&).

Variable names are always translated to uppercase.

IOS3270 handles both types of variables that are used within CMS. The first type, referred to as EXEC variables, are used by the CMS EXEC interpreter. The length of the names and values of these variables is restricted to eight (8) characters. A value always consists of a single token.

The second type of variable is used by the EXEC 2 interpreter, introduced with VM/SP Release 1, and the System Product Interpreter (REXX), introduced with VM/SP Release 3. VM/SP Release 2 introduces an interface for this type of variable, referred to as the *shared variable interface*, accessible through the subcommand EXECCOMM. IOS3270 cannot handle variables whose names or values exceed 160 characters. Names that are longer are simply not found, and values are truncated.

Variables are substituted under control of the .C function, except when they are found as parameters for function characters. In this case, substitution is always performed. However, substitution of these variables is restricted to one token only. If, for example, the variable **&FILE** has the value TEST FILE A1, only the value **TEST** is substituted. Exceptions to this are the functions .V and .&vname. For EXEC, all user variables and the internal EXEC variables &0-&30 may be referenced but not &INDEX, &RETCODE, &EXEC, &GLOBALn, and so on. Compound substitution is not performed; for example, if &A=X, &X=1 and &A1=B, &A&X will be displayed as X1 and not as B, as the EXEC interpreter would do.

Variables are recognized:

- As a separate field, for example:

  ```
  %==> &var1 %<==
  ```

- In an input field, for example:

  ```
  #3&var1 ¬&var2
  ```

- In a selector light-pen field, for example:

  ```
  ! &attn  !?&select
  ```

- As parameters for function characters
- As (part of) an intensified field, for example:

  ```
  %&var1-&var2.&var3
  ```

In the above examples, several ways to end a variable are shown:

- A blank delimiter
- A period (.) which acts as a place holder for output alignment. (Default for input fields)
- An ampersand (&) which starts a next variable (output only)
- A dash (-) to chain fields.

A variable that has no value or has a null value does not take a place on the screen, except when it ends with a period (.). Dashes (-) in chained fields are replaced by blanks ( ), unless the field is defined with attribute type Dynamic.

4.8 Using Variables in Input Fields

The following rules apply to variables used in input fields:

- Only one variable can be used in an input field.

  Note: & and - cannot be used in names in input fields. Output alignment is done automatically. The place holding character (.) should not be used.

- The default length of the input field generated is eight (8) characters if EXEC variables are used. For REXX and EXEC 2 variables, the length is equal to the length of the variable name, including the leading &. A length denominator preceding the & may be used to explicitly define the length.

  Examples:

  ```
  ¬&vname   - length =  6 (or 8 for EXEC)
  ¬3&vname  - length =  3
  ¬45&vname - length = 45
  ```

- No shifting of input fields is done (panel input/output alignment).
- A variable overlays the next field if its value is longer than its name (EXEC only).
- If a variable does not exist (is unassigned), it is created.
- If periods (.) are to be used in variable names (for REXX compound symbols), use the .J function to redefine the default place-holding character.
- Function character **N** controls editing of data in input fields, defined as a variable name as follows:

  .N not specified

– Uppercase translation of value
– Leading/trailing blanks removed
– One token only (data up to first blank in input field)

.N specified

– No editing performed, allowing for leading blanks, multiple tokens, and lowercase characters.

4.9 IOS3270 Variables

There are three variables which are automatically set by IOS3270 if called from an EXEC:

&IOSK Contains a value indicating the last key used before exit:

ENTER The ENTER key was pressed.

PA1 PA1 key was pressed. (And the PA1 EXIT option has been specified.)

PA2 PA2 key was pressed. (And the PA2 EXIT option has been specified.)

PFnn PF key nn was pressed.

PEN The selector light pen was used.

TIME The time specified with the TIME option has elapsed.

&IOSC Contains a four- or five-character decimal number representing the cursor position in the *RRCCC* form, where *RR* is the row number (00-99) and *CCC* the column number (00-999). *CCC* is 3 digits if the screen width (number of columns) exceeds 99.

&IOSD Contains the value assigned to a PF key with .F if that PF key was pressed or the first eight bytes of a selector light pen field with an **attn** designator character, causing IOS3270 to exit. If the ENTER key was used, the variable will be blank, unless it was used to define an input field and data was entered in that field.

4.10 Special Characters

There are two characters that have a special meaning if they are the first character from a PF key value or a selector light pen field with an **attn** designator character. These characters are:

) To simulate a null line (ENTER with no data). If this character is detected, processing continues. Other input on the screen is not processed.

¬ The field following this character is put in &IOSD or in the program stack, and IOS3270 returns control to the calling program. If the input was from the program stack and there was input left, it is first cleared. This function can be used with multi-panel files if immediate return to the caller is required.

4.11 Special Values In Fields

Some values have special meanings if they are used to define a PF key function or a selector light pen field with an **attn** designator character, for example:

QUIT Immediate exit RC=4 (also if entered in any input field). The stack is cleared first.

BACK Scroll back one (1) panel (not possible if input is stack).

4.12 Invoking IOS3270 with In-storage Data

Instead of feeding IOS3270 with CMS files or a CMS program stack, it is possible to dynamically create a panel in virtual storage and have it processed by IOS3270. There are two ways to present in-storage data.

Note: As IOS3270 runs in XA-toleration mode only (RMODE 24), all storage areas must reside below the 16M boundary.

Method 1

The following Assembler code is an example of the first method:

```
         .     .
DISPLAY  DS    0H
         LA    R1,IOSPLIST
         SVC   202
         DC    AL4(ERROR)
         B     .
ERROR    .     .
IOSPLIST DS    0D
         DC    CL8'IOS3270',AL4(0),AL4(IOSDATA)
         DC    CL8'other parameters and/or options'
         DC    8X'FF'
         .     .
IOSDATA  DC    AL1(L'LINE1)
LINE1    DC    C'.cf help % quit'
         DC    AL1(L'LINE2)
LINE2    DC    C'Specify input%==> ¬_________'
         .     .
         DC    AL1(L'LINEn)
LINEn    DC    C'PF1=! HELP  PF3=! QUIT'
         DC    AL1(0)
         .     .
         .     .
```

Figure 1. In-core Interface Method 1, Basic

If byte 1 of the first parameter is X'00', bytes 4 to 8 are assumed to contain a pointer to a block of virtual storage containing the input lines. Every logical record should start with a byte defining the length of the text to follow. If a length byte of zero (X'00') is found, this is treated as an end-of-file indicator.

Calling IOS3270 as described above allows for the easy modification of existing panels, but input fields are returned in the CMS program stack (if .D and .W are not used). Returning input fields to virtual storage is not only faster and easier than reading the

CMS program stack, but is in fact the only way to correctly interpret returned input fields if more than one input field is on a single line. (This restriction is removed with the VM/SP 2 version of EXEC 2.) This is also supported by IOS3270. To indicate that this way of operation is required, bytes 2 to 4 of the first parameter should contain a pointer to a 24 byte buffer. This buffer is used to return the variable values &IOSK, &IOSC and &IOSD (in this sequence). The PLIST in the previous Assembler example would then look like this:

```
         .     .
         .     .
IOSPLIST DS    0D
         DC    CL8'IOS3270'
         DC    AL1(0),AL3(IOSXLIST)
         DC    AL4(IOSDATA)
         .     .
IOSXLIST DS    0C
IOSK     DC    CL8' '
IOSC     DC    CL8' '
IOSD     DC    CL8' '
         .     .
         .     .
```

Figure 2. In-core Interface Method 1, Return of &IOSx

On return from IOS3270, virtual storage has been modified as follows:

- The fields at IOSXLIST are set as described in 4.9, “IOS3270 Variables” on page 35.
- All unused input fields contain blanks (X'40'). If the .O function was used, the first byte of the field is set to the omission character specified. Using .O seems useless in this case, though, as input entered by the operator is directly stored in the corresponding input field in the user’s virtual storage.

Note: A blank (X'40') is used as an ending delimiter for an input field. A blank can be entered by the operator. Particular care has to be paid to this, otherwise the input field will suddenly be shorter than intended. (See the .E function, which can be used in most cases to prevent this problem. See also the .J function, which allows you to define a FILL character.)

Method 2

The second in-storage interface is based on the method used by the System Product Interpreter and the EXECLOAD command, introduced with VM/SP Release 4, called File Block (FBLOCK) support. A description can be found in the *System Product Interpreter Reference, SC24-5239*, under “System Interfaces”.

The bits of the high order byte of the descriptor list length fields can be used as follows:

“xxxxxx00” Reserved, must be set to 0.

“000000x0” When on, indicates that the address points to another descriptor list. The length field specifies the length (in bytes) of the descriptor list. Acts as an imbed. Indirect addressing in combination with the skip bit can be used as a paging mechanism.

“0000000x” When on, indicates that this entry should not be processed. Can be used to selectively include or exclude lines from a panel.

To indicate that operator input is to be returned to storage, the IOSXLIST pointer must be set. The IOSDATA pointer must be set to zeros.

Note: If the input file is EXECLOADed, IOS3270 will automatically process the file from storage if the file mode is either not specified or specified as “*”. If a file mode is specified, the file will be read from disk.

Finally, a short example using the FBLOCK support:

```
         .     .
         LA    R0,EPLIST          Set EPLIST ptr
         LA    R1,IOS3270         Set PLIST ptr
         ICM   R1,B'1000',=X'01'  Set EPLIST flag
         SVC   202                Call IOS3270
         DC    AL4(1)             We handle errors
         .     .
* The extended Plist.
EPLIST   DS    0A
         DC    A(IOS3270,0,0,FBLOCK)
* The tokenized Plist.
IOS3270  DS    0D
         DC    CL8'IOS3270'
@IOSXLST DC    AL4(0)
@IOSDATA DC    AL4(0)
         DC    CL8'('
         DC    CL8'NOCLEAR'
         DC    CL8'ALARM'
         DC    8X'FF'
* The File Block.
FBLOCK   DS    0F
         DC    CL8' '  Filename
         DC    CL8' '  Filetype
         DC    CL2' '  Filemode
         DC    H'2'    Extension length in words
         DC    AL4(DLIST1,DLIST1E-DLIST1)
*        Descriptor list flags (HO length byte)
$DLSKIP  EQU   B'00000001' Skip
$DLIDA   EQU   B'00000010' Indirect descriptor list
* Descriptor list 1.
DLIST1   DS    0A
         DC    A(LINE1,L'LINE1) Data line 1
         DC    A(LINE2,L'LINE2) Data line 2
         DC    A(DLIST2)        Descriptor list 2
         DC    AL1($DLIDA),AL3(DLIST2E-DLIST2)
DLIST1E  EQU   *
* Data lines 1.
LINE1    DC    C'.cjx Set Ctl @ Col=tur Hig=rev'
LINE2    DC    C'Testing@FBLOCK support 1'
* Descriptor list 2.
DLIST2   DS    0A
         DC    A(LINE3,L'LINE3) Data line 3
         DC    A(LINE4,L'LINE4) Data line 4
DLIST2E  EQU   *
* Data lines 2.
LINE3    DC    C'.cjx Set N Co=tu Hi=re, Ctl @ Col=pu Hig=un'
LINE4    DC    C'Testing@FBLOCK support 3'
         .     .
```

Figure 3. In-core Interface Method 2

4.13 Specifying Character Attributes and Dynamic Attributes

This section describes how to define

- Character attributes
- Dynamic character attributes
- Dynamic field attributes
- Dynamic cursor positioning.

Character attributes are those assigned to a character when written into the display buffer. They are not associated with the character position. When a character is altered or deleted, the attributes associated with that character are destroyed. When a character is altered (even if overtyped with the same character) the attributes assigned to the replacing character are inherited from the field attributes.

IOS3270 allows you to assign character attributes to non-variable and variable data.

Non-variable Data

To assign character attributes to this type of data you have 2 options.

1. External from the data, providing full WYSIWYG, using .jx Set Mask as follows:

   ```
   .cjx Set Mask, Ctl @ Color=Yellow
                                                  @@@@
   The word following the arrow is yellow ===> word
   ```

   The line following the .jx Set Mask is used as a mask to parse the data line following the mask. A mask consists of Ctlchars and blanks. A blank causes field attributes to be assigned to the associated data character. A Ctlchar causes the character attributes associated with that Ctlchar to be assigned to the corresponding data character.

2. In-line with the data using .jx Set Ctl ... Scope=Character to define a pseudo field as follows:

   ```
   .cjx Set Ctl @ Color=Yellow Scope=Character
   The word following the arrow is yellow ===> @word
   ```

   Data between the Ctlchar and the delimiter will have the character attributes specified, the Ctlchar and the delimiter are removed from the input stream, thus causing a 2 position left shift.

Variable Data

.jx Set Mask cannot be used for variables as there is no input/output relation to the variable value. If a mask is specified for a variable, it is ignored. If character attributes are to be assigned to the value of a variable, the variable has to be defined as a (pseudo) field. Scope=char can be used to suppress leading and trailing blanks. The mask to be used for variable data is specified in another variable. The Dynamic attribute (.jx Set Ctl ... Dynamic) is used to indicate that a reference has to be made to such a variable, specifying

- A Ctlchar defining alternate (dynamic) field attributes
- An indication as to where the cursor has to be placed
- (A) Ctlchar(s) defining character attributes (a mask).

A reference variable is defined as an array variable with the following format (REXX notation):

```
'A'C2x(Ctl)'.$name'='Ctlf||cursor||[Ctlc||s||l]||...||[Ctlc||s||l]'
```

Ctl Is the Ctlchar defining the field (with the Dynamic attribute).

name Is the variable name used to define the field value. Only one variable name (with or without a length denominator) may be used. For example, ¬8&filename or %&mode. The format %123&fielda-&fieldb (chained fields) is not supported for fields defined with a Ctlchar having the Dynamic attribute. This is necessary in order to support the length denominator in output fields (dynamic field attribute change).

Ctlf (CL1) is a Ctlchar defining alternate field attributes (or '00'x for no change).

cursor (XL1) is the position in this field where the cursor should be placed. If '00'x, the cursor is not positioned.

Ctlc (CL1) is a Ctlchar defining character attributes to be assigned to Substring(Value(name),s,l).

s (XL1) The start of the sub-string. Defaults to 1

l (XL1) The length of the sub-string. Defaults to the rest of the field.

As a help to the user (and to allow changing the format when required), the IOSDYNAT EXEC is supplied to build and set the reference variable containing the dynamic attributes. For the calling syntax, refer to Appendix E, “The IOSDYNAT EXEC” on page 57.

It is recommended to call the IOSDYNAT EXEC to build the reference variable, even though this will degrade performance.

The variable is used to dynamically build the mask described above. The descriptors are processed left to right, each storing a sub-string of Ctlchars into the mask. Note that overlapping sub-strings containing attribute values that are not specified will inherit their value from the preceding character.

Example:

The field @12&in1 and call
Interpret iosdynat('@,in1;;;#,2,7;$,3,3;%,5,3')
builds the following mask (shown per processing step):
1: ' ####### '
2: ' #$$$### '
3: ' #$$%%%# '

The dynamic attribute support can be used from the in-storage interface (FBLOCK only) as follows. IOSDATA points to an FBLOCK-like extension area (REFBLOCK):

```
REFBLOCK DC    AL4(INPLIST,INPLISTE-INPLIST)
INPLIST  DS    0A
         DC    AL4(DYNA1,FIELD1)
         DC    AL4(DYNA2,FIELD2)
INPLISTE EQU   *
DYNA1    DS    XL(DYNFALNG+L'FIELD1)
DYNA2    DS    XL(DYNFALNG+L'FIELD2)
```

Figure 4. In-core Interface Method 2, Dynamic Attributes

The first two bytes of each field DYNA*n* are the alternate field attribute and the cursor position, the remainder is the mask, as described above.

As usual, the skip bit in the second word of the descriptor list can be used to activate or deactivate an entry.

```
*
* This DSECTs describe the format of the Refer variable as used
*  for dynamic attribute support.
*
DYNFAMAP DSECT             Dynamic field attributes
DYNFACTL DS    CL1         The Ctlchar to be used for this
                            field or '00'x for no change.
DYNFACUR DS    AL1         The absolute position in the field
                            where the cursor should be placed.
                            If 0, the cursor is not positioned.
DYNFACH  EQU   *           Start of character attribute
                            descriptor list (DYNCAMAPs)
DYNMASK  EQU   *           Start of mask for FBLOCK support
DYNFALNG EQU   *-DYNFAMAP  Size in bytes


DYNCAMAP DSECT            Dynamic character attributes
DYNCACTL DS    CL1        The Ctlchar defining character attr
DYNCASUS DS    AL1        The starting position of the substr
DYNCASUL DS    XL1        The length of the substring
DYNCALNG EQU   *-DYNCAMAP Size in bytes
```

Figure 5. DYNFAMAP DSECT

Notes:

1. The easiest format for the reference variable would have been “Ctlchar.name”.
   a. It allows an exact description of the Ctlchar/name combination.
   b. It allows a reset of all reference variables relating to one Ctlchar using the stem only.
2. The alternative was derived because REXX does not allow
   a. Setting stem.name when name is a variable. One would need to use another variable, set to “name”.
   b. characters usually used as Ctlchars in variable names.
   c. variables to start with a number.

4.14 Using The IOS3270 Options HNDEXT and HNDINT

When IOS3270 is used as a panel manager in office or database systems used by clerical personnel, the terminal will often be left in a certain state for a longer period of time, for example, displaying a selection panel. If we take an office system as an example, there may be occasions where we want to present another panel to the operator, for example to inform him or her of the arrival of a file in his/her virtual reader, or to display a message from the office system master Virtual Machine, sent with the VMCF or IUCV interface. Trapping such situations by stealing PSWs is not too difficult. The problem is in preserving what is on the screen at the moment of the interrupt, and simulating a screen

interrupt (like pressing ENTER) to let IOS3270 finish processing and return control to the caller.

The IOS3270 options, HNDEXT and HNDINT, allow you to handle the above situations. The logic behind it is as follows: Together with the option, you provide the address of a routine that should receive control when an interrupt occurs. This routine inspects the interrupt and signals through register 15 (R15) whether IOS3270 should read the data on the screen and return control to you. If you decide to handle the interrupt, you may start working on it once you regain control from IOS3270. In the interrupt-handling routine, you should not perform any action causing the screen contents to change. Preferably, you should not do anything at all except remember that you must do something on return from IOS3270. In the Assemble example below, the only action performed is the setting of a flag.

The following rules apply when the user interrupt routine gets control:

- R15 entry point address.
- R14 return address.
- R13 72 bytes save area (18 FWs).
- Storage protection is NUCLEUS key.
- The user is responsible for saving and restoring R0-R13. The save area pointed to by R13 may be used for this purpose.
- If, on return to IOS3270, R15 is 0, the screen is processed and control is returned to the caller.

The following piece of Assemble code is an example of how the options can be used:

```
         .     .
DISPLAY  DS    0H
         LA    R1,IOSPLIST    set PLIST ptr
         SVC   202            call IOS3270
         .     .
         TM    SW1,$1EXTINT   did an external int. occur?
         BO    .              br, yes
         TM    SW1,$1IOINT    did we have an I/O interrupt?
         BO    .              br, yes
         .     .
IOSPLIST DS    0D
         DC    CL8'IOS3270'   command name
         DC    AL4(IOSXLIST)  IOSX list
         DC    AL4(IOSDATA)   data list
         DC    CL8'('         option starter
         DC    CL8'HNDINT'    handle I/O interrupts
         DC    AL4(HNDIOI)    our I/O inter.handler
         DC    AL4(0)         DW filler
         DC    CL8'HNDEXT'    handle ext.interrupts
         DC    AL4(HNDEXT)    our ext.inter.handler
         DC    AL4(0)         DW filler
         DC    8X'FF'         PLIST end
         .     .
*        External interrupt handler.
         USING HNDEXT,R15
HNDEXT   .     .              examine ext. interrupt (EXTOPSW)
         BNER  R14            br, no, do not handle
         OI    SW1,$1EXTINT   handle this ext.int
         SLR   R15,R15        indicate we want control
         BR    R14            return to IOS3270
         DROP  R15            from HNDEXT
         .     .
*        I/O interrupt handler.
         USING HNDIOI,R15
HNDIOI   .     .              examine I/O inter.(CSW & IOOPSW)
         BNER  R14            br, no, do not handle
         OI    SW1,$1IOINT    handle this I/O.int
         SLR   R15,R15        indicate we want control
         BR    R14            return to IOS3270
         DROP  R15            from HNDIOI
         .     .
SW1      DS    XL1            switch byte
$1EXTINT EQU   X'80'          ext.int. to handle
$1IOINT  EQU   X'40'          I/O int. to handle
         .     .
```

Figure 6. Sample HNDEXT/HNDINT Assemble Code

Note: Due to the restructuring of CMS interrupt handing in CMS 5.5 (Bimodal CMS), the HNDEXT and HNDINT options can no longer be fully supported on this level of CMS. Refer also to the descriptions of the options.

4.15 Color Aspects

This section describes the behavior of color screens (like the IBM 3279) with respect to using the .JX functions.

Without any additional programming, the 3279 Color Display Station displays data in four different colors. The color of a field is dependent on the attributes of the field according to the following matrix:

base color (oooo)	protected	unprotected
normal intensity	blue	green
high intensity	white	red

This mode of operation is called **base color mode enabled**. The 3279 has a two-position switch, called the **base color switch**. The (oooo) position of this switch enables base color mode with the effect described above. Switching to the (oo) position disables base color mode; now there are only two colors:

base color (oo)	protected	unprotected
normal intensity	green	green
high intensity	white	white

Whenever a color other than the base color is displayed anywhere on a panel, the position of the base color switch is ignored. However, the screen behaves as if the base color switch is placed in the (oo) position!

What does this mean in practice? Suppose you have a panel with normal and intensified fields, both input and output. This shows up nicely in four colors. However, if you decide to bring some more color into this panel and, for example, change the color of one field into yellow, the colors blue and red will change to green and white.

It is important to understand that this is done by the screen and not by IOS3270. No attempt is made to simulate base color mode for those fields that will change color, because the position of the base color switch cannot be retrieved. Simulating base color mode enabled would have an opposite effect for those users that normally have the switch in the disabled position (oo).

4.16 Performance Considerations

This section discusses some points that are useful to keep in mind while using IOS3270. It is not possible to give an all-purpose solution guaranteeing good performance since there are too many factors that can influence performance. These hints are written to help the IOS3270 user avoid situations which are likely to cause performance degradation.

General

A general rule which applies to all computing processes is to find a balance between CPU and I/O load. Applied to IOS3270, this means that you must decide where you want to place your input data and in what format it should be. For example, if your input data has to be read from disk, this means overhead from CMS and CP and (usually) physical disk I/O. If disk caching is not available on your system, you may want to use EXECLOAD to place heavily used disk files in storage. If, on the other hand, you decide to use the STEM option and build all input data dynamically, this involves CPU load to build the input data and CMS overhead to retrieve the variables through the EXECCOMM interface. Whether to use one method or another (or a combination) is dependent on factors like:

- How many times the same panel will be used
- Whether the panels are static or dynamic
- How many panels are involved
- Availability of disk caching.

The preferred method of operation for DIOF-3 is with REXX. If performance is not acceptable, you may want to consider using the in-storage interface, but only as a last resort as the investment required to produce the same function will be much higher than with using REXX. Changes will require comparably higher investments.

Combining IOS3270 and EXEC Files

This section applies to very small (end user) applications, typically the ones that basically consist of one EXEC and one or a few panels.

If run from an EXEC, there is no need to have a separate file for the panel. It can be put at the end of the normal EXEC logic preceded by a label as shown in the next example:

```
  &TRACE
  &IF &N EQ 0 &GOTO -explain
  &IN = &1
-check_input
  &UPPER VAR &IN
  &IF /&IN EQ /TEST &EXIT
  &ALARM = ALARM
  &MSG = &STRING OF You should enter 'TEST', not &IN .
-explain
  IOS3270 &0 EXEC ;EXPLAIN (&NOCLEAR &ALARM
  &NOCLEAR = NOCLEAR
  &ALARM =
  &IF &IOSK NE ENTER &EXIT
  &GOTO -check_input
;EXPLAIN
.ch2
&0.----------------- Explanation --------------------- HELP
 ===> &msg
.ssc
Enter%TEST ,- press%PF1 or light pen select to exit.
%===> ¬4&in
.l24cf test
PF: 1=! test        3=! quit
```

Figure 7. Combining IOS3270 and EXEC Files

When using this technique of combined EXEC and IOS3270 files, and the file is on a disk formatted with 800-byte blocks, it is advisable to use fixed files only, to prevent performance degradation. If the file is on a disk formatted with 1K, 2K or 4K blocks, there is not much difference. To save disk space, you may very well use a file of variable format here.

If the EXEC is large, and the panel is likely to be displayed often, there is a possibility of considerable I/O overhead caused by IOS3270 searching for the start label. You may reduce this overhead by putting the panel in front of the file and by optimizing the EXEC logic by using line number references rather than labels. XEDIT's VMFOPT macro may be used to accomplish this as shown in the next example.

```
  &TRACE
*%OPTIMIZED AT 14:13:21 ON 82/05/26
*%       N O T I C E:
*% THIS STATEMENT IS NECESSARY FOR THE OPTIMIZATION PROGRAM - VMFOPT
*% DE-OPTIMIZE THIS MACRO BEFORE MAKING ANY CHANGES USING - VMFDEOPT
  &GOTO 17 -exec
;EXPLAIN
.ch2
&0.------------------ Explanation --------------------- HELP
 ===> &msg
.ssc
Enter%TEST ,- press%PF1 or light pen select to exit.
%===> ¬4&in
.l24cf test
PF: 1=! test        3=! quit
;EXEC
-exec
  &IF &N EQ 0 &GOTO 25 -explain
  &IN = &1
-check_input
  &UPPER VAR &IN
  &IF /&IN EQ /TEST &EXIT
  &ALARM = ALARM
  &MSG = &STRING OF You should enter 'TEST', not &IN .
-explain
  IOS3270 &0 EXEC ;EXPLAIN (&NOCLEAR &ALARM
  &NOCLEAR = NOCLEAR
  &ALARM =
  &IF &IOSK NE ENTER &EXIT
  &GOTO 20 -check_input
```

Figure 8. Combining IOS3270 and EXEC 2 Files

Finally, the same EXEC in REXX format:

```
/* Simple IOS3270/REXX example */
/* Start of IOS3270 panel definition
;explain
.jx Set Norm col=tur, High col=Yel typ=(hig),
.jx Set Ctl | col=red typ=(hig), Ctl ! typ=(det tab hig)
.jx Set Ctl ¬ col=gre hig=und typ=(unp null skip)
.mch
&fn.------------------ Explanation --------------------- HELP
 |&msg
.cnjx Set High col=whi
%===> ¬4&in
.sc
Enter%TEST ,- press%PF1 or light pen (CURSR) select to exit.
.l-1cf test % quit
PF: 1=! test        3=! quit
;exec    End of IOS3270 panel definition */
 Trace N
 Parse Source . . fn .
 out=0
 Parse Value '' With noclear alarm
 Address Command
 Do n=1 While ¬out
   'IOS3270' fn 'EXEC ;EXPLAIN (' noclear alarm
   noclear='NOCLEAR'
   alarm=''
   If iosk¬='ENTER' Then out=1
   If in¬='' Then Do
     chk=in
     Upper chk
     If chk='TEST'
     Then out=1
     Else Do
       alarm='ALARM'
       msg='Please enter "TEST", not' in'.'
       End
     End
   End n
 Exit
```

Figure 9. Combining IOS3270 and REXX Files

Note that the panel is at the start of the EXEC, enclosed by REXX comment delimiters. If the EXEC is EXECLOADed, IOS3270 will automatically process the in-storage version.

Note: Disk I/O can be avoided completely by using the STEM option. The IOSIVP4, IOSIVP5, or IOSIVP6 EXECs can be used as a base for this technique. For details, refer to Appendix F, “The IOSIVP6 EXEC” on page 61.

Using the STEM option is also recommended when IOS3270 is called from EXECs compiled with the VM REXX Compiler for CMS (5664-390).

Using Many Panels

If IOS3270 is used to write an application program or an online information (Help) system, it is likely that many panels are to be generated. All these panels must be stored somewhere and retrieved by IOS3270 when the information is needed. There are several ways to put all the panels on disk. A separate file can be made for every panel, which amounts to a large number of files in the end.

Another way is to put them (either grouped by function or not) into one large file and use the IOS3270 start/end label facility to selectively retrieve a panel. Working this way creates a lot of I/O overhead if the file becomes large. The CMS system has a solution in the form of the MACLIB and TXTLIB facilities. To take advantage of the IOS3270 label processing, a modified version of the CMS MACLIB program, called IOSLIB, is distributed with the IOS3270 package. It is discussed in Appendix C.

As from Version 3, IOSLIBs are created with variable records (RECFM=V). This will in most cases reduce the amount of disk space required to store the library. However, it requires IOSLIBs to be stored on CMS minidisks **not** formatted with BLKSIZE=800 to reach an acceptable level of performance.

It is difficult to define when to use separate files or a large file with label indexes or an IOSLIB. By experimenting and keeping in mind the points discussed above, the user will soon discover what suits his or her needs best.

Appendix A. Questions from IOS3270 Users

1. Does IOS3270 run in Bimodal-CMS?
   - Yes, but in toleration-mode (RMODE 24, AMODE 24) only.
2. Why doesn't IOS3270 substitute variable names containing a period (.)?
   - It does, but you should use the .J function (position 8) to respecify the variable place holder character.
3. How do I get more than one token back in a variable?
   - Use the .N function.
4. I used .N and now my input fields are truncated at the first blank on re-display.
   - Use the .E function to redefine the ending delimiter.
5. How do I get mixed case input in a variable?
   - Use the .N function.
6. I have a 3278 model 3 but IOS3270 only uses 24 lines. How can I use all 32 lines?
   - Use the .M function.
7. Why do I lose all my input data using a PF key?
   - Use the .Y function.
8. I only have a simple 3277 screen, should I use .J or .JX?
   - Use .JX where possible. .JX works on all screen types. When you get a newer screen, you will be ready for it.
9. Can I use the .I function to imbed in-storage data?
   - No.
10. Not all input data from the panel is returned in the program stack.
    - You probably have more than 130 characters of input. Use the .X function.

Appendix B. Summary of Function Characters

The following is a summary of all the available function characters. This summary can be referenced by entering **IOS3270 ?**.

.A	Sounds audible alarm whenever this panel is displayed.
.B	Next data line is a bottom title.
.C	Next lines may contain field definition characters (Ctlchars).
.D	Defines or resets secondary input/output.
.E	Defines the field ending delimiter.
.F	Sets PF key functions.
.H	Displays next line(s) intensified.
.I	Imbeds.
.J	Redefines field definition characters.
.JX	Extended define.
.K	Saves input fields in keep file.
.L	Skips to a specified line on the screen.
.M	Maximum screen size to be used.
.N	Do not edit input field variable values.
.O	Defines an omission character.
.P	Page (panel) eject.
.Q	Like .R, but allows input and/or selector light pen fields to be defined.
.R	Reads a record from secondary input.
.S	Spaces a line.
.T	Defines or resets top title(s).
.V	Stores variable values.
.W	Writes all input entered up to this line to a file, defined with .D.
.X	Puts all input entered up to this line in the program stack.
.Y	Processes input fields when a PF key is pressed.
.Z	Clears input fields on bottom two lines.
.&name	Substitutes this line by the value of symbol *name*.

Appendix C. The IOSLIB Program

The IOSLIB program is a modified version of the CMS MACLIB program (from VM/370 Release 3). An online explanation of the command syntax and the functions available is provided by entering **IOSLIB ?**. Modifications are in the following areas:

- Creating libraries with RECFM V to reduce disk space. In addition, the complete dictionary is written as one record. To reach an acceptable level of performance, IOSLIBs should **not** be stored on 800B formatted disks.

  Old IOSLIB files (RECFM F) for which no source is available can be converted to RECFM V using the IOSLIB COMP function.

- Creating dictionary entries for IOS3270 labels found in input file(s), thus reducing I/O overhead.

- Adding RET function to retrieve members from a library.

- Allowing input file(s) to be RECFM V, thus reducing disk space needed to store input files.

For further details on the CMS MACLIB command, refer to the *IBM VM/SP CMS Command and Macro Reference Manual*, SC19-6209, or the *IBM VM/XA SP CMS Command Reference Manual*, SC23-0354.

A suggested procedure for creating IOSLIBs is to have all members in one large file containing *COPY statements in front of each member. If a member is to be updated, edit the file and update it. Then use the IOSLIB GEN function to create a new library. This seems to be a most effective way to keep libraries up-to-date and as small as possible.

Appendix D. The GOODIES/GDCSS Package

The GOODIES/GDCSS package was developed with the following objectives:

- Allow programs that have outgrown the transient area and cannot run in the user area for one reason or another to run:
  - From DMSFREE storage
  - As a nucleus extension
  - From a discontiguous shared segment (DCSS) (called a saved segment on VM/XA SP).
- Minimize the number of shared segments by allowing several independent programs to be saved in and run from the same DCSS, combined with a bootstrap, which transfers control to the correct function.
- Relieve the programs from taking care of DCSS interfaces, DMSFREE clean-up and NUCXLOAD.

The GOODIES/GDCSS package contains all the functions necessary to generate the bootstraps, relocatable modules, and DCSS. DCSS bootstraps may also be run as nucleus extensions using NUCXLOAD with the SYSTEM option.

GOODIES refers to the DCSS containing tools, often called “goodies”. GDCSS stands for Generalized DCSS interface.

GOODIES ASSEMBLE contains the names of all functions to be part of the DCSS and, the DCSS bootstrap. GDCSS ASSEMBLE contains the code which forms the function bootstrap and is what is first executed when the desired function is called. GOODEND ASSEMBLE is just a stub to show the end of the DCSS in the LOADMAP.

The interface between CMS and a function is provided by the program GDCSS. GDCSS only exists as a TEXT deck and is executed to create a DCSS bootstrap and an eXecution MODule (XMOD). The INSTGOOD EXEC does all module generation and builds the DCSS if requested to do so. The interface bootstrap ensures that the target program receives the proper control for correct operation, be it as a DCSS, a nucleus extension, or DMSFREE storage. When necessary, the bootstrap interface automatically loads the relocatable (ADCON free) program into DMSFREE storage. At program termination, the bootstrap does all necessary clean-up before returning to the caller. The target program need never concern itself with which mode of operation is being used.

To install IOS3270, you have the following options:

DMSFREE Storage

Just place the IOS3270 MODULE and XMOD file, generated by the INSTGOOD EXEC, on the support disk. Specify **no DCSS name** and **N** to the questions asked. The interface module will not find the DCSS and loads the XMOD into DMSFREE storage, permitting IOS3270 to run.

DCSS (or Saved Segment)

On VM/SP systems: Re-SYSGEN your VM system, setting aside a DCSS to hold IOS3270. You may use any name you wish for this DCSS. The default (assigned in the INSTGOOD EXEC) is GOODIES. The starting address for this DCSS is determined by the specification provided in the DMKSNT entry.

```
GOODIES NAMESYS SYSSIZE=64K,                    *
               SYSNAME=GOODIES,                 *
               SYSSTRT=(361,1),                 *
               SYSVOL=SYSRES,                   *
               SYSPGCT=16,                      *
               SYSPGNM=1632-1647,               *
               SYSHRSG=(102),                   *
               VSYSADDR=IGNORE
```

Figure 10. Prototype Entry in DMKSNT for GOODIES

On VM/XA systems: You theoretically have 4 options.

1. Install IOS3270 in its own 1M saved segment
2. Install IOS3270 as a part of saved segment GOODIES (1M)
3. Install IOS3270 as a member using segment packing
4. Install IOS3270 as a part of member GOODIES using segment packing.

Options 1 and 2 are impractical due to the large amount of wasted storage. Option 3 would require a change in the GDCSS interface to dual path the code, depending on whether it is running on VM/SP or VM/XA, including the necessary options in INSTGOOD to specify which mode of operation is wanted (option 3 or 4). So, for practical reasons only option 4 can be used. If no other programs are to be included in GOODIES, a member size of 24K (six 4K pages) is sufficient.

Create a GOODIES member in segment space *space1* by using the DEFSEG command, for example:

```
DEFSEG GOODIES 660-666 SR SPACE space1
```

On VM/SP and VM/XA systems: Ensure that your virtual storage is large enough to hold the DCSS before running INSTGOOD. When running INSTGOOD, specify the name of this DCSS and respond **Y** to the question asking if a DCSS should be generated. Copy the created MODULE and XMOD to your support disk. The interface module uses the DCSS unless virtual storage is too large, in which case the XMOD is loaded into DMSFREE storage and executed.

To run the bootstrap as a nucleus extension, the user must issue the command NUCXLOAD IOS3270 (SYSTEM. This is independent of whether the DCSS or XMOD is eventually used. If the DCSS cannot be used, the bootstrap loads the XMOD into DMSFREE TYPE=NUCLEUS storage and replaces itself by the XMOD, thus making it the nucleus extension for this function.

To include other modules in the DCSS, modify GOODIES ASSEMBLE and run INSTGOOD to include them. Functions included must be relocatable (ADCON free) and not larger than 64K. This limit is assumed by both the INSTGOOD EXEC and the GDCSS bootstrap interface. INSTGOOD simply copies the second record from the GENMODed function and calls it **function XMOD**. GDCSS reads only one record into DMSFREE storage. This technique was devised for programs which had outgrown the 8K CMS transient area. Since XMODs are placed into DMSFREE storage, a free storage problem could easily result if the function programs were permitted to be any size. Consequently, a 64K limit was chosen. It is relatively simple to change INSTGOOD and GDCSS to increase these limits, if desired.

Remember that the GDCSS interface gives all modules a NUCLEUS protect key when they receive control.

The following information is provided for those who wish to write programs to take advantage of the GDCSS interface or to better understand how this interface operates.

GDCSS permits any program which is reentrant and ADCON free to run in DMSFREE storage, in a DCSS, or as a nucleus extension. All programs are run with a nucleus protect key.

In addition, GDCSS can be used to interface to programs only in a DCSS, only as nucleus extensions, or only to be run in DMSFREE storage. Although maximum flexibility is obtained if the program can run in either way, the following are supported in any desired combination:

- DCSS: reentrant
- Nucleus Extension: relocatable (ADCON free) and serially reusable
- DMSFREE storage: relocatable (ADCON free).

As long as the program observes the restrictions for the environment in which it is to run, it need not do anything to interface with GDCSS.

GDCSS itself runs either from the CMS transient area or as a nucleus extension and is always GENMODed with the SYSTEM option.

A function consists of a bootstrap and an ADCON free XMOD. The bootstrap is called ***function* MODULE** (for example, *function* = IOS3270). This module contains the function name and, if specified during creation, the DCSS name. Calling the function gives control to the bootstrap which transfers control to the bootstrap in the DCSS, which in turn transfers control to the correct function. Return is made to the function bootstrap, if clean-up is needed, or directly to the caller, following normal CMS conventions.

If the DCSS cannot be attached, or if it was not specified during creation, the bootstrap loads the XMOD into DMSFREE storage and transfers control thereto.

When the DCSS or XMOD terminates, control initially comes back to the bootstrap, which ensures that proper clean-up is done before returning control to the caller. For a DCSS, the bootstrap will do a PURGESYS only for those invocations where it had to do a LOADSYS and if that has been specified during creation of the bootstrap. For an XMOD, the bootstrap will DMSFRET the storage into which the XMOD was loaded. In either case, the bootstrap also releases any dynamic storage needed for its own use before returning to the caller.

When used as a nucleus extension, the bootstrap is all that is really loaded into nucleus-free storage. If the bootstrap determines that the DCSS cannot be utilized, it replaces itself with the XMOD and is no longer used. The current implementation requires that the end user issues the NUCXLOAD command to load these functions as nucleus extensions. One possible way of doing this is to NUCXLOAD these functions with an EXEC that is executed by all users during CMS initialization, such as the SYSPROF EXEC, introduced with VM/SP Release 5. To prevent storage fragmentation at a later stage, it may be a good idea to include a call to IOS3270 here in case the XMOD needs to be loaded. A way to load IOS3270 without generating error messages or have the screen cleared is by using a simple IOS3270 (VERSION call.

The various interfaces are accomplished as follows:

DCSS

This interface is independent of how GDCSS was loaded. GDCSS checks for the existence of the specified DCSS, and if necessary, loads it. Control is transferred to the interface module at the start of the DCSS (GOODIES), which then finds the specified routine and transfers control to it. If the DCSS was not loaded, return is made directly to the caller. If the DCSS could not be used for any reason, GDCSS checks to see if it is transient or a nucleus extension and continues appropriately. If the DCSS had to be loaded, GDCSS sets up a tail-end processing routine to PURGESYS the DCSS on completion, if specified during creation. On systems with many users, it will usually be cheaper to keep the DCSS attached. This mode of operation is the default.

Nucleus Extension and no DCSS

R2 points to the NXBLOCK. GDCSS loads the XMOD into DMSFREE TYPE=NUCLEUS storage and then updates the NXBLOCK to indicate that this copy of the specified command is the nucleus extension. GDCSS then releases its own storage and is no longer involved in future calls to this function.

The following register content is required for programs to run as nucleus extensions:

R0 As set by caller, points to new form PLIST

R1 As set by caller, points to old form PLIST

R2 A(NXBLOCK)

R12 Entry address

R13 A(general register save area)

R14 Return address

R15 Entry address

R3-R11 Are UNPREDICTABLE

DMSFREE execution

If GDCSS cannot use a DCSS and is not itself a nucleus extension, then it loads the relocatable XMOD into DMSFREE storage. It also places a small “tail” of its own code into DMSFREE storage and provides this as the return address. When return is made, the tail section of GDCSS does appropriate clean-up, including the release of the DMSFREE storage.

The actual address of the function in the DCSS is contained in the DCSS bootstrap (GOODIES ASSEMBLE). This means that a change in one of the programs does not affect any other program as long as the DCSS is re-saved. The INSTGOOD EXEC creates the function bootstraps each time, but these only have to be replaced on the system disk if GDCSS has been changed. You can change the function code and, if you have not changed GDCSS, the old interface bootstrap module still works perfectly.

Similarly, if you change only GDCSS, you just have to replace the interface bootstrap module on the system disk. The XMOD does not have to be replaced, nor does the DCSS have to be saved. An option has been added to INSTGOOD to permit creation of the interface bootstrap without requiring the creation of the corresponding XMOD.

Appendix E. The IOSDYNAT EXEC

The IOSDYNAT EXEC shown below can be used to dynamically change the attributes of fields and for cursor positioning. It is included here as reference material to explain the calling syntax.

```
/** eupdate: Do not modify this file. It was built by the EUPDATE procedure.
  * eupdate: The source file was: IOSDYNAT 0EXEC A1
  * eupdate: Built by Theo W.H. Alkema - 10001880 at UITHONE
  * eupdate: Date: 05/19/87 - Time: 16:55:42 - DST */
/* Set dynamic field and/or character attributes and position cursor */
/* Calling format:
    Interpret iosdynat(Arg1; <Arg2>; <Arg3>; <Arg4 <... ;Argn>>

     Arg1 defines the affected field(s)
      Format: Parm1 <,Parm2>
       Parm1: Ctlchar
              Ctlchar is the Ctlchar with which this field is defined.
       Parm2: name
              Specifies the variable name following the Ctlchar.
              The name may be preceded by a length denominator.

     Arg2 defines a dynamic field attribute or a function
      Format: <Parm1>
       Parm1: Ctlchar | 'RESET' | 'DROP'
              Ctlchar defines the Ctlchar to be used for this field.
                      Defaults to Ctlchar from Arg(1).
              RESET   specifies to assign the null value to the
                      variable(s) containing the dynamic attributes as
                      defined by Arg(1).
              DROP    specifies to drop the variable(s) containing the
                      dynamic attributes as defined by Arg(1).

     Arg3 defines the cursor position within the field
      Format: <Parm1>
       Parm1: Cursor position; absolute location in this field.
              Do not specify if the cursor should not be positioned.

     Arg4-n define character attributes
      Format: Parm1 <... <;Parmn>>
       Parmx: Ctlchar <,start <,length>>
              Ctlchar specifies the Ctlchar to be used. The extended
                      attributes associated with this Ctlchar are applied
                      to the substring defined by start and length.
              start   defines the absolute starting position of the
                      substring within the field. Defaults to 1.
              length  defines the length of the substring. Defaults to
                      the remainder of the field.
  */
```

Figure 11 (Part 1 of 4). IOSDYNAT EXEC

```
iosdynat:
 maxl=160
 Parse Arg input
 Parse Value input with arg1 ';' arg2 ';' arg3 ';' arg4
 Parse Value('') With stem suffix val
 /* Need at least 2 arguments */
 If arg1='' Then Signal noargs
 If (arg2='' & arg3='' & arg4='') Then Signal noargs

 /* Process arg1 */
 Parse Value arg1 With ctl ',' name
 /* Check the Ctlchar */
 If Length(ctl)¬=1 Then Signal invctl
 stem='A'C2x(ctl)'.'
 If name¬='' Then suffix='$'name

 /* Process arg2 */
 ctl=arg2
 /* Check for functions */
 If Length(ctl)>1 Then Do
   Upper ctl
   Select
     When ctl='RESET' Then Return stem||suffix"=''"
     When ctl='DROP'  Then Return "Drop" stem||suffix
     Otherwise Signal invfunc
     End
   End
 If ctl='' Then ctl='00'x
 val=ctl

 /* To continue we need a field name */
 If name='' Then Signal noname

 /* Process arg3 */
 d='00'x
 If arg3¬='' Then Call cknum arg3
 val=val||d
```

Figure 11 (Part 2 of 4). IOSDYNAT EXEC

```
 /* Process arg4 */
 /* Parse the character attribute definition */
 Do While arg4¬=''
   Parse Value arg4 With parm ';' arg4
   Parse Value parm With ctl ',' start ',' lng .
   If ctl='' Then Leave
   If Length(ctl)¬=1 Then Signal invctl
   If start='' Then d='01'x
   Else Call cknum start
   val=val||ctl||d
   If lng='' Then d='00'x
   Else Call cknum lng
   val=val||d
   End

 /* Finally, check & return the assignment */
 If Length(val)>maxl Then Signal toomany
 Return stem||suffix'="'val'"'

cknum:
 Arg d
 If Datatype(d)¬='NUM' Then Signal notnum
 If d<1   Then Signal toosmall
 If d>maxl Then Signal toomuch
 d=D2c(d)
 Return
```

Figure 11 (Part 3 of 4). IOSDYNAT EXEC

```
noargs:
 Say 'Insufficient number of arguments given'
 Exit
noname:
 Say 'No field name specified'
 Exit
invctl:
 Say 'Invalid Ctlchar:'''ctl''''
 Exit
notnum:
 Say 'Numeric value expected:'''d''''
 Exit
toosmall:
 Say 'Minimum value is 1:'''d''''
 Exit
toomuch:
 Say 'Maximum value('maxl') exceeded:'''d''''
 Exit
toomany:
 Say 'Maximum range('maxl') exceeded'
 Exit
invfunc:
 Say 'Unknown function:' ctl
 Exit
```

Figure 11 (Part 4 of 4). IOSDYNAT EXEC

Appendix F. The IOSIVP6 EXEC

The IOSIVP6 EXEC below is included as an example of how the STEM, FROM, and FOR options can effectively be used. It is included here as reference material only. Similar programs are available as IOSIVP4 and IOSIVP5 EXEC. Basically, the three are slight variations on a similar theme, only the input source is different, or differently handled. These programs can be used (after sufficient tailoring) by applications requiring their functions.

```
/* Basic functional test/demo of the following IOS3270 options:
    STEM    stem    - ala EXECIO
    FROM    m       - start at stem||m
    FOR     n       - process n records
    VERSION         - query version

 - General notes:
   1.This code handles the dynamic model where only the first panel
     is dynamically build before presenting it. Higher levels are
     dynamically build when selected. Once build, they are retained.
     Panel nesting is done on row (line) basis.
     A static model (where all information is pre-generated and
     available in an IOSLIB for example, typically HELP menus)
     is not yet available. The main differences with the dynamic
     model are:
      a.the data has to be put into variables first (EXECIO (STEM)??
      b.come to think of it, why shouldn&csq.t you use standard HELP????
 - Application notes:
   1.Keep it SIMPLE by creating logical arrays which you
     imbed selectively, using FROM and FOR options where needed.
   2.This procedure handles &*q.&$CONT
floating" pages. A page can start
     on any line (by using the cursor point function '/').
     It also automatically adjusts the page size to the number
     of rows available on the used screen.
     It also keeps track of "lines" versus "records", meaning
     it recognizes whether a logical record results in a physically
     displayed line on a panel.
     The logic could be considerably simplified by using fixed
     pages but then it would, of course, not be general. All in
     all, the method used here does not appear to be very costly
     and handles all situations.
     A possible problem arises scrolling the screen to a point
     where attributes have been changed from the default (as
     defined in the header section) when the control records that
     set the new attributes are not included in the body part.
     This code does not contain logic to keep track of that
     situation. For now we expect the user to point at the start
     of a logical section on a panel.
```

Figure 12 (Part 1 of 9). IOSIVP6 EXEC

```
 - Performance notes:
   1.Avoid variable substitution by IOS3270 when possible.
     As the panels are build dynamically anyway, let REXX
     do the work, it is better equipped for that.
     Note: IOS3270 issues an SVC 202 (EXECCOMM) for each variable!
 */

 Address COMMAND
 Call init                              /* initialize               */
 Call bld_pnl                           /* build main panel lines   */
 Call bld_hdr                           /* build header lines       */
 Call bld_body                          /* build body lines         */
 Call bld_bot                           /* build bottom lines       */
 Call display                           /* get it displayed         */
 Return                                 /* all done                 */

display:  /* Display panels */
 Do Until done
   stem=stem.lvl
   for=rows2for(body_cnt,start.lvl)    /* get # of body itnos      */
   If (for¬=0 & d.stem.0>start.lvl+for-1) Then more='more ...'
   Else more=''
   /* describe the body part */
   pnl.2='.ci (stem d.stem. from' start.lvl 'for' for
   'IOS3270 (STEM PNL.' alarm clear iosc.lvl
   If rc¬=0 Then Leave
   cur_row=Substr(iosc,1,2)            /* save the cursor row      */
   cur_col=Substr(iosc,3)              /* save the cursor column   */
   iosc.lvl=iosc                       /* save the cursor position */
   clear='NOCLEAR'                     /* don't clear next round   */
   alarm=                              /* reset possible alarm     */
   msg=''                              /* clear possible msg       */
   Select                              /* determine what to do     */
     When iosk='ENTER' Then Call pr_enter
     When Substr(iosk,1,2)='PF' Then Call pr_pfk
     Otherwise Nop
     End
   Call pr_input
   End
 Return
```

Figure 12 (Part 2 of 9). IOSIVP6 EXEC

```
pr_pfk:                             /* PFkey pressed           */
 Select
   When iosd='N' Then Call foreward
   When iosd='P' Then Call backward
   When iosd='S' Then Call lvl_select
   When iosd='E' Then Call lvl_end
   When iosd='/' Then Call point
   When iosd='?' Then Call retrieve
   When iosd='H' Then Call help
   Otherwise done=1
   End
 Return

pr_enter:                           /* ENTER key pressed       */
 If on_body() Then Call lvl_next    /* if on body line select  */
 Else iosc.lvl=inpc.
 Return

lvl_select:                         /* SELECT key pressed      */
 If on_body() Then Call lvl_next    /* if on body line select  */
 Else call invcur
 Return

lvl_next:
 If Verify(Substr(d.stem.t,1,1),'|','M') Then Do/* if more levels*/
   h2.lvl=hdr.2                       /* save header             */
   Parse Value Substr(d.stem.t,2) With hdr.2 ' more' ./* new ... */
   hdr.2='From' hdr.2                 /*   ...   header          */
   lvl=lvl+1                          /* bump level count        */
   stem.lvl=stem'.'t                  /* set new stem            */
   stem=stem'.'t                      /* set new stem            */
   start.lvl=1                        /* set new start           */
   iosc.lvl=''                        /* set cursor              */
 If Symbol('d.stem.0')='LIT' Then Call create_lines
                                      /* create next level info  */
   End
 Else Call emsg 'No further details available'
 Return

lvl_end:                            /* END function            */
 lvl=lvl-1                          /* bump level count        */
 If lvl=0 Then done=1               /* if not last level       */
 hdr.2=h2.lvl                       /* restore header          */
 Return
```

Figure 12 (Part 3 of 9). IOSIVP6 EXEC

```
foreward:                           /* FORWARD function         */
 n=row2recf(body_cnt,start.lvl+1)  /* get itno of next page    */
 If n¬=start.lvl Then Do           /* if not same as current   */
   If n<start.lvl Then Call emsg 'Wrap around from last page'
   start.lvl=n                     /* set new start            */
   iosc.lvl=''                     /* and position cursor      */
   End
 Else Call emsg 'This is the last page'
 Return

backward:                           /* BACKWARD function        */
 n=row2recf(-body_cnt,start.lvl-1) /* get itno of previous pg  */
 If n¬=start.lvl Then Do           /* if we can go back        */
   start.lvl=n                     /* restore start itno       */
   iosc.lvl=''                     /* set cursor               */
   End
 Else Call emsg 'This is the first page'
 Return

point:                              /* Cursor scroll function   */
 If on_body() Then Do              /* If on body line then     */
   t=row2recf(-1,t)                /* get 1st record number    */
   start.lvl=t                     /* start there              */
   iosc.lvl='03'cur_col            /* set cursor (leave column)*/
   End
 Else Call invcur
 Return

retrieve:                           /* Display previous command */
 inp_rr=
 inp=inp.inpr#
 iosc.lvl=inpc.inpr#
 inpr#=inpr#-1
 If inpr#=0 Then inpr#=inp.0
 If inp.inpr#='' Then inpr#=inp#-1
 If inpr#=0 Then inpr#=1
 Return
```

Figure 12 (Part 4 of 9). IOSIVP6 EXEC

```
pr_input:
 If iosd='?' Then Return
 If inp¬='' Then Select          /* command available        */
   When inp='?' Then Call retrieve
   When inp=inp_rr Then Do       /* reset command in error   */
     inp_rr=
     inp=
     iosc.lvl=
     End
   Otherwise Call pr_cmd         /* process the command     */
   End
 Return

pr_cmd:                         /* process a command        */
 inp_rr=
 If inp¬=inp.inp# Then Do       /* save the command         */
   inp.inp#=inp
   inpc.inp#=iosc
   inpr#=inp#
   inp#=inp#+1
   If inp#>inp.0 Then inp#=1
   End
 Trace O                        /* execute the command      */
 If Substr(inp,1,1)='&' Then Address CMS Substr(inp,2)
 Else Address CMS inp
 Trace N
 If rc¬=0 Then Do
   If rc=-3 Then msg='Unknown Command'
   Else msg='Return code' rc
   alarm='ALARM'
   inp_rr=inp
   iosc.lvl=iosc
   End
 Else Do
   If Substr(inp,1,1)¬='&' Then Do
     inp=''
     iosc.lvl=inpc.
     End
   End
 Return

on_body: /* Determine whether the cursor is on a body line */
 t=cur_row-2                            /* convert row to body line */
 If (t>0 & t<=body_cnt) Then Do         /* if pointing in body part */
   t=row2rec(t,start.lvl)               /* convert row to record    */
   If t¬=0 Then Return 1                /* ok if within body part   */
   End
 Return 0
```

Figure 12 (Part 5 of 9). IOSIVP6 EXEC

```
invcur:
 Call emsg 'Cursor position not valid for requested operation'
 Return

help: /* provide cursor dependent help information */
 Select
   When cur_row=1 Then Call emsg 'You are pointing at the header line'
   When cur_row=2 Then Call emsg 'You are pointing at the message line'
   When cur_row=rows Then Call emsg 'You are pointing at the PFkey line'
   When cur_row=rows-1 Then Call emsg ,
                      'You can execute any CP/CMS command from this line'
   Otherwise Do
     If on_body() Then Do
       If Verify(Substr(d.stem.t,1,1),'|','M')
       Then Call emsg 'Press the ENTER/PF2 key to display more',
                       'information about this item'
       Else Call emsg 'There is nothing more to say about this item'
       End
     Else Call emsg 'This is just a blank line, nothing more'
     End
   End
 Return

emsg:
 msg=Arg(1)
 alarm='ALARM'
 Return

rows2for: /*rows, start
  rows  = the number of rows (within the body)
          For example, 3 meaning 3 body lines.
  start = the starting record number
  Determine the number of records to be processed to display
  'rows' rows of data, starting at record 'start'.
  */
 v=row2rec(Arg(1), Arg(2))
 If v=0 Then Return d.stem.0-Arg(2)+1
 Return v-Arg(2)+1
```

Figure 12 (Part 6 of 9). IOSIVP6 EXEC

```
row2recf: /*row, start
  row   = the displayed data row number (within the body)
          For example, 3 meaning the 3rd body line.
  start = the starting record number
  Convert a row number to first record number.
  The last qualifier of the current array of variables is
  what we call a record number here.
  The first record of row n is considered to be the record
  following row n-1. This may be an IOS3270 control record
  setting color, for example.
  */
 v=row2rec(Arg(1), Arg(2)-1)
 If v¬<d.stem.0 Then Return start.lvl
 Return v+1

row2rec: /* row, start
  Convert a row number to a record number.
  (compensates IOS3270 control records and null assignments)
  Required for paging and record count calculations when data
  records (displayed lines) are intermixed with IOS3270 control
  records.
  */
 Arg target
 If target<0 Then Do
   incr=-1
   stop=1
   target=Abs(target)
   End
 Else Do
   incr=1
   stop=d.stem.0
   target=target
   End
 u=0
 Do v=Arg(2) To stop By incr
   If Symbol('d.stem.v')='LIT' Then Iterate v
   If Substr(d.stem.v,1,1)='.' Then Iterate v
   u=u+1
   If u=target Then Return v
   End v
 Return 0

bld_pnl: /* Build the panel lines */
 pnl.1='.i (stem hdr.'
 pnl.2='.ci (stem d.stem.'
 pnl.3='.i (stem bot.'
 pnl.0=3
 Return
```

Figure 12 (Part 7 of 9). IOSIVP6 EXEC

```
bld_hdr: /* Build the header lines */
 hdr.1='.mnyhjx Set Norm Col=whi Ali=att comp'
 hdr.2='Use the cursor TAB and ENTER/PF2 keys to display selective entries'
 hdr.3='.j &&&&&&&'
 hdr.4='.chjx Set Norm Col=tur, High Col=red, Ctl   Col=yel'
 hdr.5='&msg'
 hdr.6='.e jx Set Ctl | Typ=tab'
 hdr.0=6
 Return

bld_bot: /* Build the bottom lines */
 bot.1='.l-2f H S E % / ? P N % % % Q'
 bot.2='.ch2jx Set Hig Col=whi, Ctl | hig=und col=gre typ=unp'
 bot.3='===>|65&inp                  ',
          '                                                &more',
 bot.4='.jx Set Mask'
 bot.5='                                                     ' ,
        '                          '
 bot.6='PF 1=Help 2=Select 3=End          5=/ 6=? 7=Prev 8=Next',
        '                12=Quit'
 bot.0=6
 Return

bld_body: /* Build arrays of body lines */
 lvls=9
 lines=34
 Call create_lines
 Return

create_lines:
 n=0
 Do r=1 To lines By lvl
   n=n+1
   If (lvl<lvls & n//(lvls-lvl+1)=0)
   Then d.stem.r='| Info level 'lvl 'record' r 'line' n '    more ...'
   Else d.stem.r=' Info level 'lvl 'record' r 'line' n
   End r
 d.stem.0=r-lvl
 Return
```

Figure 12 (Part 8 of 9). IOSIVP6 EXEC

```
init: /* Initialize for first round */
 'IOS3270 (VERSION' /* iosd='IOS3270 V2.1 R4.101.01 07/13/88' */
 Parse Value iosd With . 'R' . '.' iosd '.' .
 If (rc=1 | iosd<101) Then Do
   Say 'Sorry, your IOS3270 is outdated'
   Exit 4
   End
 rows=C2d(Substr(Diag(8c),5,2))  /* get number of screen rows*/
 body_cnt=rows-4                 /* number of body lines     */
 alarm=
 stem=
 stem.=
 iosc.=
 clear=
 lvl=1
 start.=1
 done=0
 inp=                            /* input ....               */
 inp.=                           /*            areas         */
 inp.0=10                        /* # of retrieve buffers    */
 inp#=1                          /* 1st input area           */
 inpr#=1                         /* 1st retrieve buffer      */
 inpc.=Right(rows-1,2,'0')||07   /* cursor on command line   */
 Return
```

Figure 12 (Part 9 of 9). IOSIVP6 EXEC

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

Printed in Denmark