Screen Definition Facility II for VSE

Run-Time Services

Release 6

SH12-6312-00

Screen Definition Facility II for VSE

IBM

Run-Time Services

Release 6

SH12-6312-00

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

First Edition (December 1997)

This edition applies to Release 6 Modification Level 0 of Screen Definition Facility II for VSE, Program Number 5746-XXT, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Vienna Software
SDF Development/Service
Dept 00/174
Lassallestrasse 1
A–1020 Vienna
Austria
Europe

E-mail: kpesen@vnet.ibm.com

If you prefer to send comments by fax, use this number:

Your International Access Code + 43 + 1 + 21145–4490

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Notices

Trademarks and service marks

About this book

This book explains how to use the Screen Definition Facility II (SDF II) dialog management services to develop dialogs for use in application prototypes for the SDF II target systems—CICS/BMS, CSP/AD, VisualGen, ISPF, IMS/MFS, and GDDM–IMD—and to develop application prototypes of those application programs.

Who should use this book?

This book is for application programmers who develop dialogs and application prototypes using the SDF II dialog manager (SDF2/DM). Users should be familiar with coding in REXX in the VSE environment.

Syntax conventions used in this book

Syntax shown in this book follows a simplified Backus Naur form. Syntax diagrams that might otherwise be over-complex and hard to understand quickly are separated into their major constituent parts.

[ ]	Square brackets identify an optional syntactic unit, which you may use.
|	The logical OR sign indicates that you can choose the syntactic unit to its left or right.
...	An ellipsis indicates that the immediately preceding syntactic unit is repeated.
COMMAND	Text shown in roman boldface font is a command or keyword.
variable	Text shown in italic font represents a variable.

What you will find inside

Chapter 1	Explains how to create an application prototype that uses the panel definitions, keylists, messages, and file-tailoring skeletons you define. It also explains how to use dialog variables and variable services, how to use the file-tailoring services of SDF2/DM to read skeleton files and write tailored output that can be used to drive other functions, and how to use table services to use and maintain sets of dialog variables.
Chapter 2	Describes the sections of a panel definition.
Chapter 3	Describes the panel processing statements.
Chapter 4	Describes the control variables.
Chapter 5	Explains how to establish keylists for your panels.
Chapter 6	Describes how to write messages to be displayed on a panel.
Chapter 7	Explains how to define file-tailoring skeletons.
Chapter 8	Describes, in alphabetical order, the dialog management services of SDF II.

Chapter 9	Lists and explains the system commands that users can issue from within a dialog.
Chapter 10	Lists and describes the system variables, relating each to the dialog management service to which it applies.
Appendix	Describes how to specify special characters, lists of variables, and double-byte character set (DBCS) strings.
Glossary of terms and abbreviations	Contains a list of terms and their definitions used in SDF II.
SDF II publications	Contains a list of related publications.

At the end of the book is an index.

Chapter 1. Introduction to the SDF II dialog manager

The SDF II dialog manager (SDF2/DM) provides a set of services to be used in dialog applications. These applications can include panels that conform to CUA guidelines. SDF2/DM services are used by SDF II customizable user exits and the application prototype facility. This chapter provides an overview of how to write applications and application prototypes that use the SDF2/DM services.

The following groups of services are available:

- Display services
- Variable services
- Table services
- Message services
- File-tailoring services

For detailed information about specific SDF2/DM dialog management services, see Chapter 8, “Dialog management services” on page 132.

Preparing an application prototype

An application prototype created in the SDF II dialogs is a simple SDF2/DM application that essentially displays a series of panels. To make your application prototype more realistic, or to quickly create simple full-screen applications, you can add logic and SDF2/DM service calls to the generated application prototype EXEC.

An SDF II VSE application prototype is started and controlled by the SDF2 transaction. It runs in a dynamic or static VSE partition.

This section explains how to prepare an application prototype that uses the panel definitions, keylists, messages, and file-tailoring skeletons that you have defined.

Setting up the application prototype

For SDF2/DM, the panel, keylist, message, and file-tailoring skeleton objects must be stored in VSE libraries. The object libraries can be added to the VSE // LIBDEF SOURCE,SEARCH= statement in the DGIPROC.PROC procedure.

To retrieve an object from a specific sublibrary, rather than from the search chain, use the SDF2/DM LIBDEF service. DLBL and EXTENT information for all libraries referenced by SDF2/DM must be available in either the DGIPROC.PROC procedure or the standard labels.

For information on how to customize DGIPROC, refer to *SDF II Administrator’s Guide*.

If the application prototype uses the profile pool to save information across sessions, the profile is written to the library specified in the LIBDEF PHASE,CATALOG statement of the DGIPROC.PROC procedure. The member name of the profile file is *userid*.PROFDGI1 (default) or *userid*.PROF*xxxx*, where *xxxx* are the first four characters of the *applid* specified in the CONTROL OPEN APPLID or SELECT NEWAPPL service.

Starting the application prototype

To run your application prototype, SDF II provides the SDF2 transaction, which starts your REXX procedure or program. You can tailor the job submit exit (DGIJOBFS) and DGIPROC.PROC to meet your installation requirements.

To start your application prototype, enter:

SDF2 command format: start application prototype

SDF2 [*request*] [**(** [*options*]]

request is:

[{ **CMD(***exec* [*parm*]) |
exec [*parm*] |
PANEL(*panel***)** [**OPT(***menu-option***)**] |
PGM(*program***)** [**PARM(***parm***)**] }]

[**NEWAPPL(***appl***)**]

options is:

[[**ACCOUNT=***account*]
[**JOBCLASS=***jobclass*]
[**LANG=***language*]
[**LSTCLASS=***list-class*]
[**LSTDISP=***list-disposition*]
[**OPTION=***menu-option*]
[**PUNCLASS=***punch-class*]
[**PUNDISP=***punch-disposition*]
[**PURGE**]
[**PWD=***password*]
[**SPEC=***specification*]
[**TOUSER=***to-userid*]
[**USER=***userid*]
[**WAIT=***wait*]]

exec	The name of a REXX EXEC. The // LIBDEF PROC search chain is searched for *exec*.PROC.
parm	Parameters for the REXX EXEC or program.
panel	The name of a selection panel. The // LIBDEF SOURCE search chain is searched for *panel*.A.
menu-option	An option for a selection panel, such as **1** or **2.3**.
program	The name of the program. The // LIBDEF PHASE search chain is searched for *program*.PHASE.
appl	The application ID used for a new profile pool. It can be 1 to 4 characters long.
ACCOUNT=*account*	The job account information.
JOBCLASS=*jobclass*	The job class.

LANGUAGE=*language*	The 3-character language identifier of the national language to be used, such as ENU for U.S. English. The default language is determined from the library search order, which you set in DGIPROC.PROC.
LSTCLASS=*list-class*	The class for print output.
LSTDISP=*list-disposition*	The disposition for print output.
OPTION=*menu-option*	The parameters for the OPTION JCL statement separated by commas (no blanks are allowed).
PUNCLASS=*punch-class*	The class for punch output.
PUNDISP=*punch-disposition*	The disposition for punch output.
PURGE	Delete the joblog and printed output at the end of the session.
PWD=*password*	The password.
SPEC=*specification*	A customizable specification. In this field, you can specify a string of up to 8 characters for use in the start-up procedure.
TOUSER=*to-userid*	The destination user ID for print output.
USER=*userid*	The user ID.
WAIT=*wait*	The timeout value, in seconds, to limit waiting for connection to the batch partition. The range is 1 through 9999; the default is 120.

Invoking dialog management services

Dialog management services can be invoked as follows:

- From a REXX program, using a command. Because REXX handles variables differently than do other languages, some of the SDF2/DM variable services do not apply to REXX programs.
- From a program written in another language, using a call. You can call dialog management services from any language that uses standard OS register conventions for call interfaces, and the standard convention for signaling the end of a variable length parameter list. Assembler programs must include code to implement the standard save area convention.

  You might do this if you want to convert an application prototype written in REXX to PL/I, for example.

Invoking a service with a REXX command

Within a REXX program, you invoke a dialog management service as follows:

Syntax

DGIEXEC *service-name parameter ...*

DGIEXEC must be specified in uppercase. You can specify parameters in any order. Many parameters consist of a keyword followed by a value in parentheses.

If you specify the same keyword more than once with different values, the last occurrence is used.

Prefix each dialog call with the ADDRESS command or select the SDF2/DM host environment with ADDRESS DGIEXEC.

Syntax

ADDRESS DGIEXEC *service-name parameter* ...
ADDRESS DGIEXEC *service-name parameter* ...

or

ADDRESS DGIEXEC
DGIEXEC *service-name parameter* ...
DGIEXEC *service-name parameter* ...

For some services, you specify a list of variable names. For example, the syntax for the VGET service is:

Syntax

DGIEXEC VGET *name-list* [**PROFILE**]

In this case, *name-list* is a list of up to 254 dialog variable names, separated by commas or blanks. If *name-list* consists of more than one name, it must be enclosed in parentheses. Parentheses are optional if the list consists of only one name. For example:

```
DGIEXEC  VGET  (AAA,BBB,CCC)
DGIEXEC  VGET  (LNAME FNAME I)
DGIEXEC  VGET  (XYZ)
DGIEXEC  VGET  XYZ
```

The last two lines are equivalent.

Invoking a service with a program call

Within a program in a language other than REXX, you use DGILINK (the SDF2/DM subroutine interface) to invoke a dialog management service.

To obtain the address, use LOAD (CDLOAD) DGILINK. Base the DGILINK call on that address.

The following rules apply to DGILINK calls:

- All parameters are positional: you must specify the parameters in the order shown.
- To accept the default value for an optional parameter, you can specify one or more blanks within single quotes. For example: `'  '`
- To accept the default value for every remaining parameter (when every remaining parameter is optional), you can simply stop providing values.

- You must ensure that a high-order 1 bit is generated after the end of the parameter list. In PL/I and COBOL, this is done automatically. In Assembler, you use the VL keyword of the CALL macro. In C, you use the PRAGMA statement, as follows:

  ```
  #pragma linkage(DGILINK,OS)
  ```

In Chapter 8, "Dialog management services" on page 132, call statements are shown in PL/I syntax. Service names and keyword values are shown as literals, enclosed in single quotes ('). For example:

```
CALL  DGILINK ('DISPLAY ', 'MENU    ');
```

You can use variables instead of literals. This is required if you are using a language such as COBOL, which does not allow literals within a call statement.

For example, you can use variables in PL/I as follows:

```
DECLARE  SERVICE CHAR(8) INIT('DISPLAY '),
         PANEL   CHAR(8) INIT('MENU    '),

:
CALL  DGILINK (SERVICE, PANEL);
```

You can use variables in COBOL as follows:

```
WORKING-STORAGE SECTION.
   77  SERVIS      PICTURE A(8) VALUE 'DISPLAY '.
   77  PANEL       PICTURE A(8) VALUE 'MENU    '.

:
PROCEDURE DIVISION.
   CALL 'DGILINK' USING  SERVIS PANEL.
```

You can use variables in C as follows:

```
#include
#pragma linkage(DGILINK,OS)
#define SERVICE "'DISPLAY '"
#define PANEL   "'MENU    '"
main()
  DGILINK(SERVICE,PANEL);
```

You can also use assignment statements for variables. For example:

```
SERVICE='DISPLAY';              (PL/I)
MOVE 'DISPLAY' TO SERVIS.       (COBOL)
strcpy(SERVICE,"DISPLAY");      (C)
```

Declaration statements for PL/I

In PL/I programs, the following declare statements should be included:

```
DECLARE DGILINK   /*NAME OF ENTRY POINT*/
        ENTRY
        EXTERNAL  /*EXTERNAL ROUTINE*/
        OPTIONS(  /*NEEDED OPTIONS*/
        ASM,      /*DO NOT USE PL/I DOPE VECTORS*/
        INTER,    /*INTERRUPTS*/
        RETCODE); /*EXPECT A RETURN CODE*/
```

Parameter types for the DGILINK call

The following types of parameters can appear in a calling sequence to DGILINK:

Service name or keyword
A left-justified character string that must be coded as shown in the description of the particular service. The string can be up to 8 characters long. It is recommended that all service names and keywords be padded with blanks to their maximum length of 8 characters.

Single name
A left-justified character string. If the string is less than the maximum length for the particular parameter, it *must* have a trailing blank to delimit the end of the string. The maximum length for most names is 8 characters.

Numeric value
A fullword fixed binary number.

Numeric name
A dialog variable in which a number is stored. If these variables are defined in a program module, they can be either fullword fixed binary variables or character string variables. If the values returned are character, they are right-justified with leading zeros.

Name list–string format
A list of dialog variable names coded as a character string. Each name is 1 to 8 characters. The string must start with a left parenthesis and end with a right parenthesis. Within the parentheses, the names can be separated with commas or blanks. For example:

```
'(AAA BBB CCC)'
```

When the list consists of a single name, the parentheses are not required. If parentheses are not used, a trailing blank is required if the name is less than 8 characters in length. A maximum of 254 names can be listed in the string format.

Name list–structure format
A list of dialog variable names passed in a structure. Each name is 1 to 8 characters long. The structure must contain the following information in the following order:

Count	Fullword fixed binary integer containing the number of names in the list.
Reserved	Fullword fixed binary integer that *must* contain a value of 0.
List of names	Each element in the list must be an 8-byte character string. Within each element, the name of the variable must be left-justified with trailing blanks. The maximum number of names in the list is 254.

Commands and function keys

Within your dialog, a user can issue commands in any of the following ways:

- Type a command on the command line and press Enter.
- Press a function key that corresponds to a command. The value of the function key can be set by the application developer in the keylist, or by the user with the KEYS command.
- Type a value on the command line, then press the appropriate function key. The command line value is taken as the parameter string for the requested function.
- Select a function from an action-bar pull-down.

If the command is an SDF2/DM system command, SDF2/DM takes the appropriate action. If the command is not an SDF2/DM system command, SDF2/DM passes it on to the application prototype.

For more information about SDF2/DM system commands, see Chapter 9, "System commands" on page 217.

Return codes from services

Each service returns one of the following return codes:

0	Normal completion. The service was completed without errors.
4, 8	Exception condition. Something occurred that is not necessarily an error, but that the dialog should know about. The dialog can take action based on the return code.
12, 16, 20	Error condition. The service could not be completed because of errors. If you specify CONTROL ERRORS RETURN, control is returned to the dialog, and system variable ZERRLM contains the long message associated with the return code. Otherwise, the panel shown in Figure 1 on page 8 is displayed.

For a REXX command, the code is returned in the procedure variable RC. For a program call, the code is returned in register 15.

Programs coded in PL/I can examine the return code by using the PLIRETV built-in function. The following declare statements are required:

```
DECLARE DGILINK EXTERNAL ENTRY OPTIONS(ASM INTER RETCODE);
DECLARE PLIRETV BUILTIN;
```

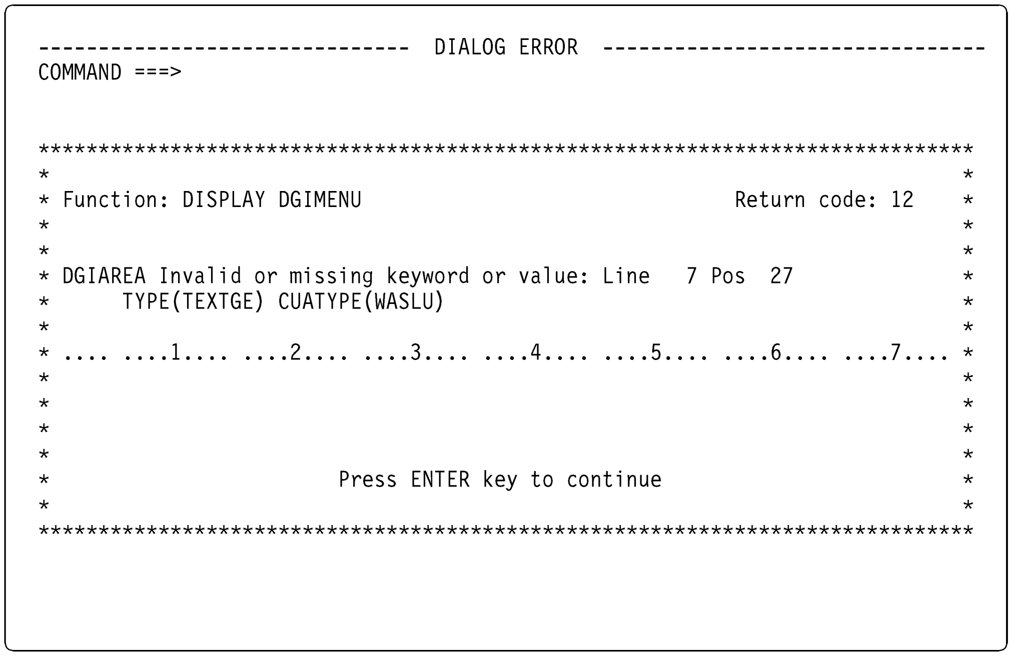
Programs coded in COBOL can examine the return code by using the RETURN-CODE built-in variable.

In C, return codes are passed back as the function value of DGILINK.

Diagnostic information

The panel shown in Figure 1 is displayed when a dialog management service returns a return code of 12 or higher. Specify CONTROL ERRORS RETURN to override this panel and return control to the application prototype.

The panel identifies the failing function, including the first four parameters, the return code, and the type of error. For a syntax error in a panel definition, the line that contains the error is also shown.



```
------------------------------  DIALOG ERROR  -------------------------------
COMMAND ===>


******************************************************************************
*                                                                            *
* Function: DISPLAY DGIMENU                              Return code: 12    *
*                                                                            *
*                                                                            *
* DGIAREA Invalid or missing keyword or value: Line   7 Pos  27             *
*      TYPE(TEXTGE) CUATYPE(WASLU)                                           *
*                                                                            *
* .... ....1.... ....2.... ....3.... ....4.... ....5.... ....6.... ....7.... *
*                                                                            *
*                                                                            *
*                                                                            *
*                                                                            *
*                        Press ENTER key to continue                         *
*                                                                            *
******************************************************************************
```

Figure 1. Example error recovery panel

If you respond with NO, the function pool is reset. In any case, the return code is passed to the application prototype.

Using the display services

The display services enable a dialog to display information and to interpret responses from users. The display services are:

DISPLAY	Display a panel, or redisplay the current panel.
ADDPOP	Add a pop-up window. Until the pop-up is removed by the REMPOP service, the DISPLAY service causes a panel to be displayed within the pop-up.
REMPOP	Remove a pop-up window or remove all pop-up windows.
SETMSG	Display a message on the next panel.
GETMSG	Get information about a message without displaying it.
PQUERY	Get information about a dynamic area on a panel.
HELP	Display a help panel under the control of the application prototype.

LIBDEF	Use a second library, in addition to the default library, for panels, keylists, and messages.
CONTROL	Change the way the next panel displays, or specify how errors should be handled.

Panel processing considerations

When the DISPLAY service is invoked from an application prototype, the panel name, message ID, cursor field, and cursor position can be specified. If no panel name is specified, the current panel is redisplayed.

After the panel has been displayed, the user can enter information and press the Enter key. All input fields are automatically stored in dialog variables of the same name, and the)PROC section of the panel definition is then processed. If any condition occurs that causes a message to be displayed (such as a verification failure, a MSG=*value* condition in a TRANS, or an explicit setting of .MSG), processing continues to the)HELP or)END section. The)REINIT section is then processed if it is present. The panel is then redisplayed with the first message that was encountered.

When the user again presses the Enter key, all input fields are stored and the)PROC section is again processed. This sequence continues until the entire)PROC section has been processed without any message conditions being encountered. The panel display service then returns, with a return code of 0, to the application prototype that invoked it.

When a panel is displayed, the user can enter a **cancel**, **end**, **exit**, or **return** command. If the input fields are not in a scrollable area, they are stored and the)PROC section is processed but no message is displayed, even if a MSG condition is encountered. The panel display service then returns to the application prototype with a return code of 8. In scrollable areas, only the input fields that have been displayed will be stored.

Using variable services

To communicate data between the dialog management services and the dialog elements, SDF2/DM uses dialog variables. A dialog variable's value is a character string of 0 to 32 KB. Some services restrict the length of dialog variable data.

Dialog variables are referred to symbolically, by a name of 1–8 characters. Alphanumeric characters A–Z, 0–9, #, $, and @ can be used in the name, but the first character cannot be numeric.

Dialog variables can be used within panels, messages, skeleton definitions, and dialog functions. For example, a dialog variable name can be defined in a panel definition, then referred to in a function of the same dialog. Alternatively, the variable can be defined in a function, then used in a panel definition to initialize information on a display panel, then later used to store data entered by the user on the display panel.

Dialog variables are stored in variable pools.

These pools and the types of dialog variables that reside in them are as follows:

Function pool Contains variables accessible only by that function. A variable that resides in the function pool of the function currently in control is called a *function variable*.

Shared pool Contains variables accessible only by dialogs belonging to the same application prototype. A variable that resides in the shared pool of the current application prototype is called a *shared variable*.

Global pool Contains variables accessible by all application prototypes. A variable that resides in the global pool is called a *global variable*.

Profile pool Contains variables that are automatically retained for the user from one session to another. A variable that resides in the profile pool is called an *application profile variable* or *profile variable*. Profile variables are automatically available when an application prototype begins and are automatically saved when it ends.

The number of shared, global, function, and profile variables that can exist at any one time depends on the amount of storage available.

Using variable services in REXX

If your application prototype is written in REXX, variables are handled as follows:

- Variables referred to in a panel definition are stored in REXX variables.
- Within SDF2/DM, system variables (whose names start with the letter Z) are stored in the shared pool. You can also copy REXX variables to the global or shared pool. This lets you pass variable values from one application prototype to another.
- You can permanently store the values of system variables and other variables in a profile pool. This lets you keep the same value of a variable between SDF2/DM sessions.

You can thus have up to four versions of the same variable at the same time. These versions do not necessarily have the same value.

Within your REXX application, an assignment statement changes only the value of the REXX variable. To update the value of a variable in the shared pool, global pool, or profile pool, use the VGET, VPUT, and VERASE dialog management services.

Within the)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC sections of a panel definition, you can update the values of REXX variables. You can also use the VGET and VPUT statements (which are related to the VGET and VPUT services) to update values in the shared pool or the profile pool. When the DISPLAY service searches for the value of a variable, it checks first for a REXX variable, then for a value in the shared pool, then for a value in the profile pool.

Note: System variables are listed in Chapter 10, “System variables” on page 224. Some of them (indicated by “non” in the tables) have values that you cannot modify in your application prototype. For example, you cannot modify ZSTDDATE, which contains the current date. To avoid confusion with system variables, do not use a name that starts with Z for any variable that you define yourself.

VGET service

To copy the current value of a variable from the shared pool to a REXX variable, use the VGET service with the SHARED keyword. This is illustrated in Figure 2.

```
DGIEXEC VGET (ZPFSHOW ABC) SHARED
```

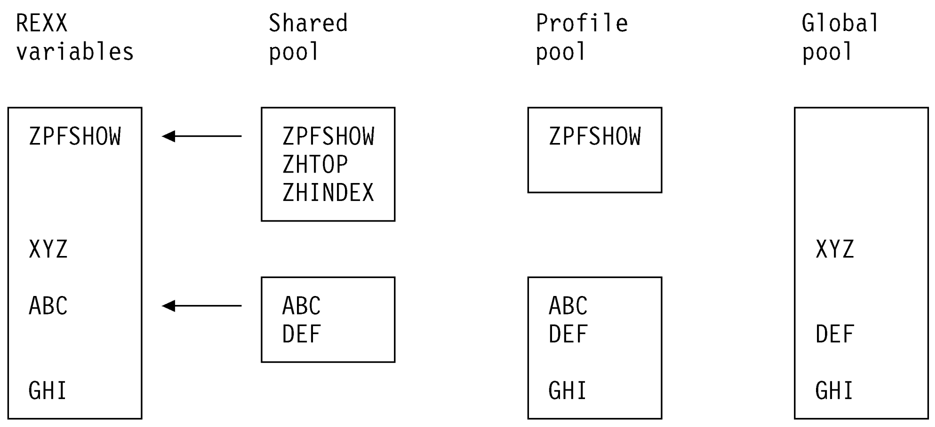


Figure 2. VGET service with SHARED keyword (REXX)

To copy the current value of a variable from the profile pool to a REXX variable, use the VGET service with the PROFILE keyword. If the variable also exists in the shared pool, the value in the shared pool is not changed. This is illustrated in Figure 3.

```
DGIEXEC VGET (ZPFSHOW ABC GHI) PROFILE
```

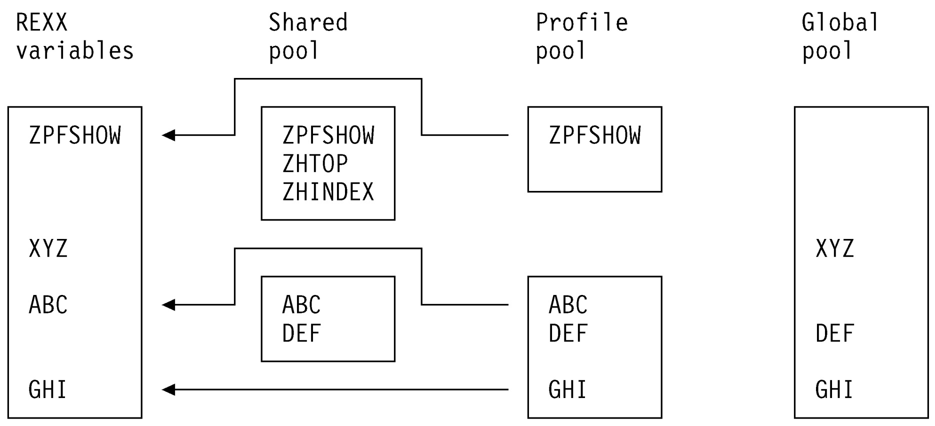


Figure 3. VGET service with PROFILE keyword (REXX)

To copy the current value of a variable from the global pool to a REXX variable, use the VGET service with the GLOBAL keyword. If the variable also exists in the shared pool, the value in the shared pool is not changed. This is illustrated in Figure 4.

```
DGIEXEC VGET (XYZ DEF) GLOBAL
```

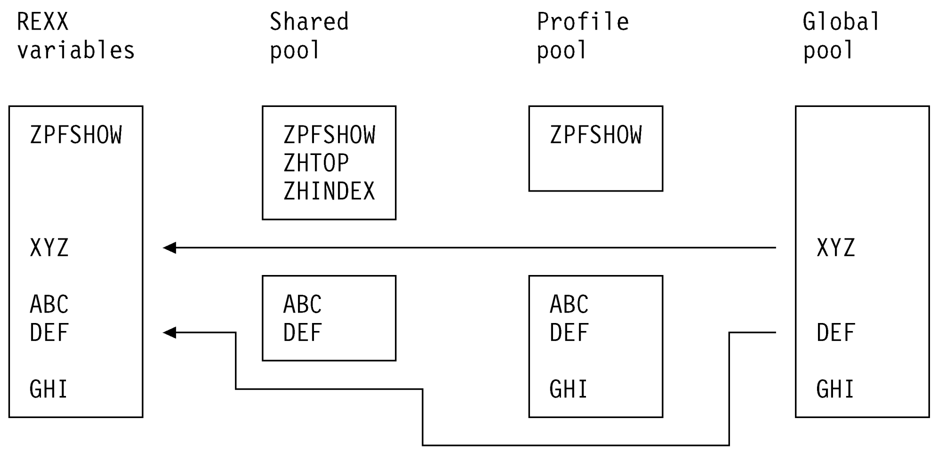


Figure 4. VGET service with GLOBAL keyword (REXX)

If you specify VGET with the ASIS keyword or with no keyword, the value in the shared pool (if any) is copied. If the variable does not exist in the shared pool, the value in the profile pool is copied.

VPUT service

To copy the current value of a REXX variable to the shared pool, use the VPUT service with the SHARED keyword. This is illustrated in Figure 5.

```
DGIEXEC VPUT (ZPFSHOW ABC) SHARED
```

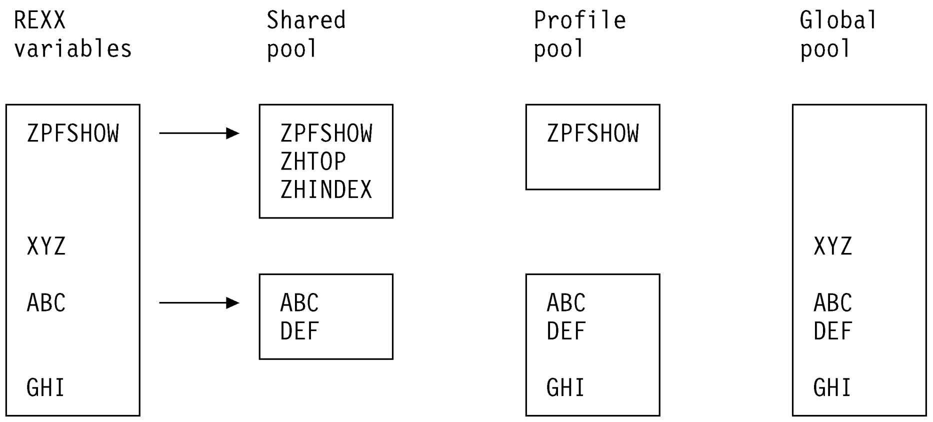


Figure 5. VPUT service with SHARED keyword (REXX)

To copy the current value of a REXX variable to the profile pool, use the VPUT service with the PROFILE keyword. Any value in the shared pool is not changed. If no REXX variable with that name exists, the current value in the shared pool is copied to the profile pool. This is illustrated in Figure 6.

```
DGIEXEC VPUT (ZPFSHOW ZHTOP DEF GHI) PROFILE
```

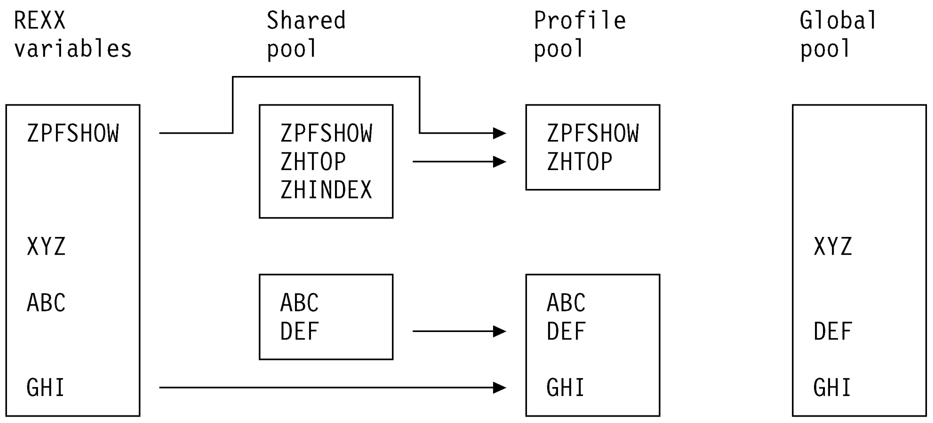


Figure 6. VPUT service with PROFILE keyword (REXX)

To copy the current value of a REXX variable to the global pool, use the VPUT service with the GLOBAL keyword. Any value in the shared pool is not changed. If no REXX variable with that name exists, the current value in the shared pool is copied to the global pool. This is illustrated in Figure 7.

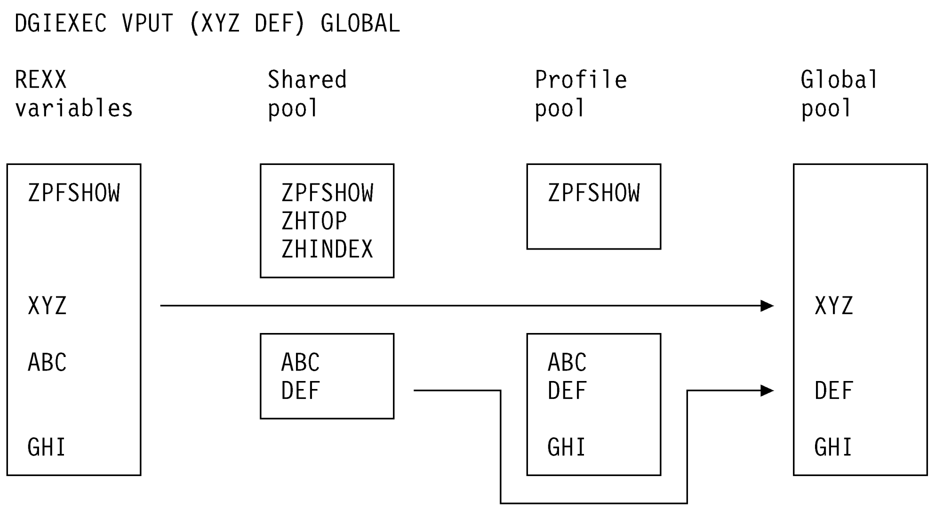


Figure 7. VPUT service with GLOBAL keyword (REXX)

If you specify VPUT with the ASIS keyword or with no keyword, the value in the shared pool (if any) is updated. If the variable does not already exist in the shared pool, the value in the profile pool (if any) is updated. If the variable exists in neither the shared pool nor the profile pool, a new variable is created in the shared pool.

VERASE service

To erase a variable from the shared pool, use the VERASE service with the SHARED keyword. This is illustrated in Figure 8.

```
DGIEXEC VERASE (ABC) SHARED
```

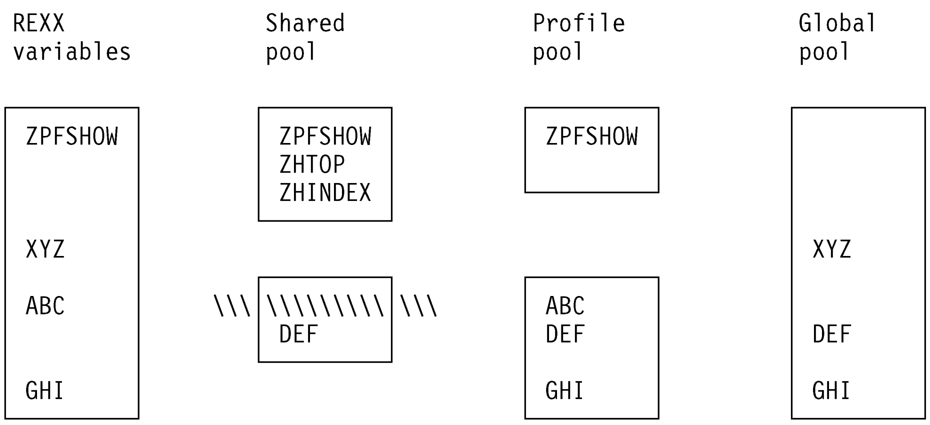


Figure 8. VERASE service with SHARED keyword (REXX). Erasure of the variable is indicated by the \ characters.

To erase a variable from the profile pool, use the VERASE service with the PROFILE keyword. This is illustrated in Figure 9.

```
DGIEXEC VERASE (ZPFSHOW DEF) PROFILE
```

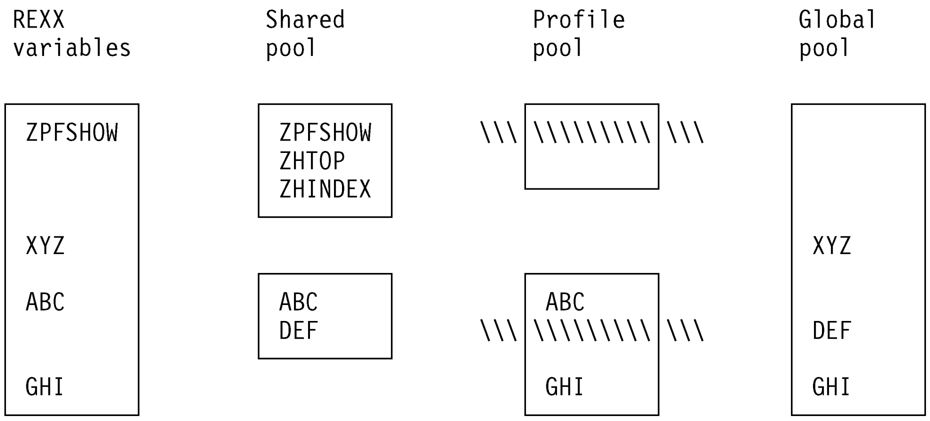


Figure 9. VERASE service with PROFILE keyword (REXX). Erasure of the variable is indicated by the \ characters.

To erase a variable from both the shared pool and the profile pool, use the VERASE service with the BOTH keyword.

If you specify VERASE with the ASIS keyword or with no keyword, the value in the shared pool (if any) is erased. If the variable does not exist in the shared pool, the variable is erased from the profile pool instead.

Using variable services in a language other than REXX

If your application prototype is written in a language other than REXX, variables are handled as follows:

- Your application prototype can manage variables under its own control in user storage.
- Variables referred to in a panel definition are stored in a function pool. You can also use the VDEFINE service to create a variable at an arbitrary location in user storage, with a pointer to the variable in the function pool.
- Within SDF2/DM, system variables (whose names start with the letter Z) are stored in the shared pool. You can also copy other variables to the shared pool.
- You can store variables in the global pool. This lets you pass variable values from one program or application to another.
- You can permanently store the values of variables in a profile pool. This lets you retain the value of a variable across SDF2/DM sessions.

These different versions of the same variable do not necessarily have the same value.

Within your application prototype, an assignment statement changes only the value in user storage. To update the value in the function pool, the shared pool, the global pool, or the profile pool, you use the VCOPY, VREPLACE, VPUT, and VERASE dialog management services.

Within the)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC sections of a panel definition, you can update values in the function pool. You can also use the VGET and VPUT statements, which are related to the VGET and VPUT services, to update values in the shared pool or in the profile pool. When the DISPLAY service searches for the value of a variable, it looks first in the function pool, then the shared pool, then the profile pool.

Note: System variables are listed in Chapter 10, “System variables” on page 224. Some of them (indicated by “non” in the tables) have values that you cannot modify in your application prototype. For example, you cannot modify ZDATE, which contains the current date. To avoid confusion with system variables, do not use a name that starts with Z for any variable that you define yourself.

VCOPY service

To copy the current value of a variable to user storage, use the VCOPY service. This is illustrated in Figure 10.

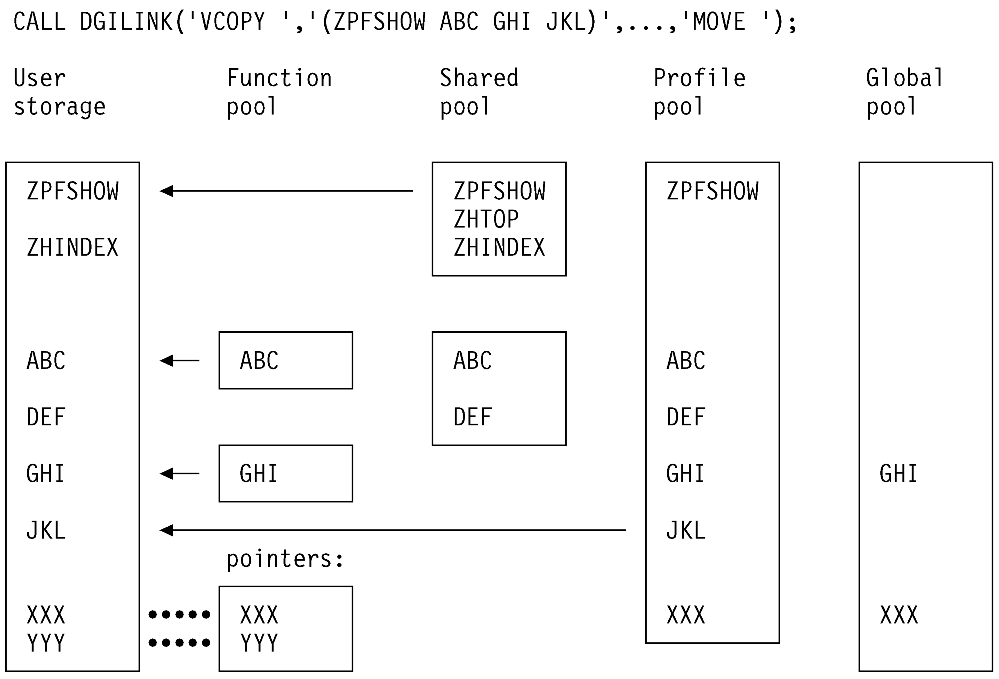


Figure 10. VCOPY service (non-REXX)

SDF2/DM looks for a value first in the function pool, then in the shared pool, then in the profile pool, and copies the first value that it finds.

You can also use the LOCATE keyword to locate a variable in the function pool or in the shared pool, but not copy it to user storage. You cannot use the LOCATE keyword to locate a variable in the profile pool.

VREPLACE service

To copy the current value of a variable in user storage, use the VREPLACE service. This is illustrated in Figure 11.

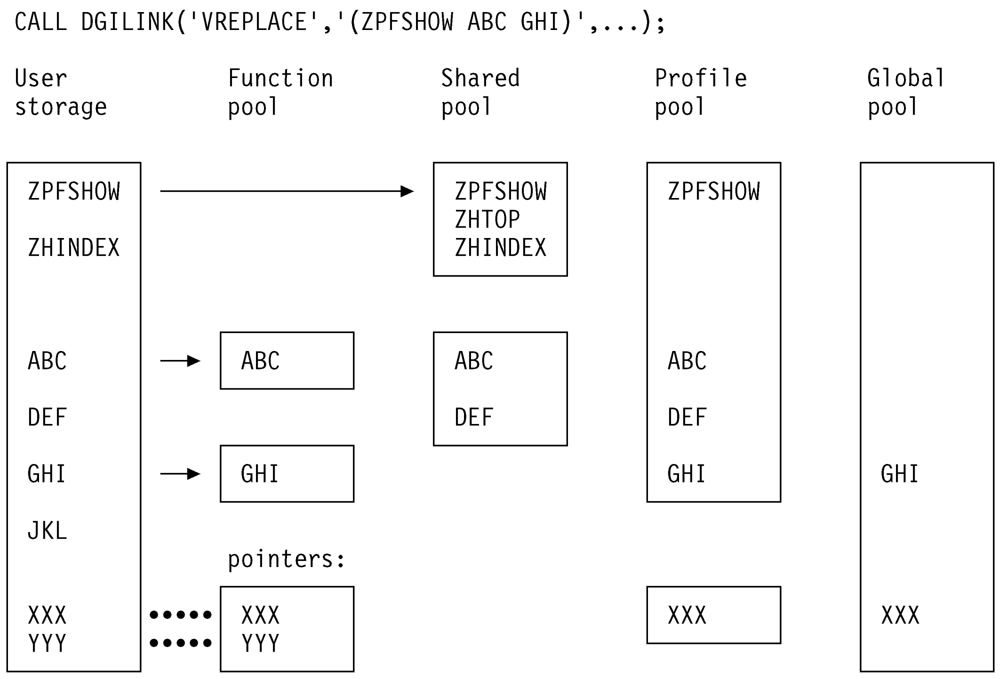
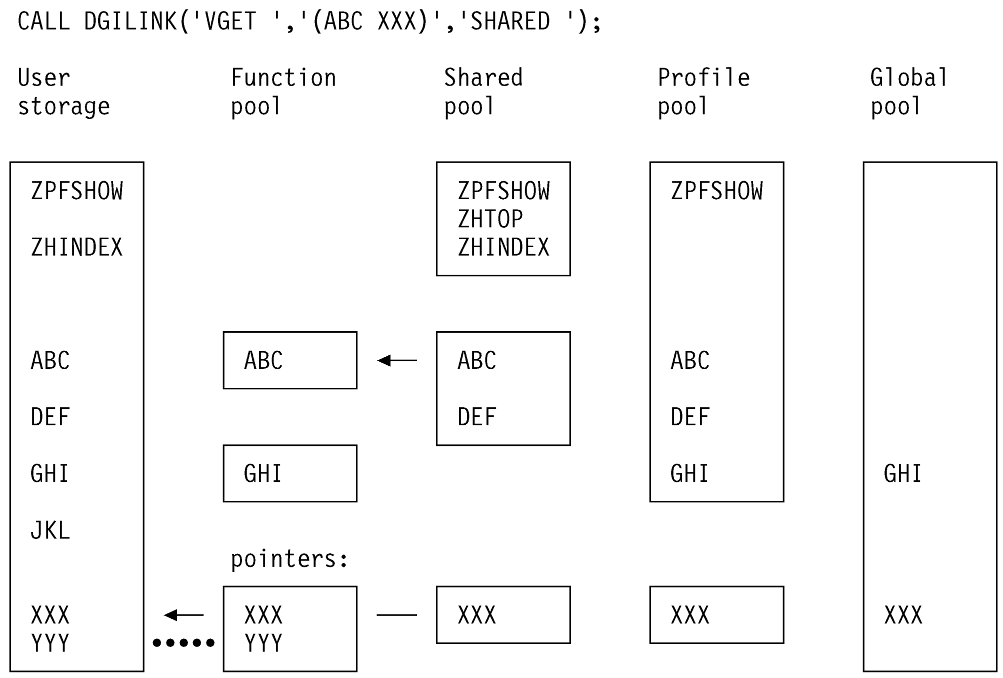


Figure 11. VREPLACE service (non-REXX)

SDF2/DM updates the value in the function pool, if any. If the variable does not exist in the function pool, SDF2/DM updates the value in the shared pool, if any. If the variable exists in neither the function pool nor the shared pool, SDF2/DM creates a new variable in the function pool.

VGET service

To copy the value of a variable from the shared pool to the function pool, use the VGET service with the SHARED keyword. If the variable was created with VDEFINE, with a pointer in the function pool, the value in user storage is updated. This is illustrated in Figure 12.



```
CALL DGILINK('VGET ','(ABC XXX)','SHARED ');
```

Figure 12. VGET service with SHARED keyword (non-REXX)

To copy the value of a variable from the profile pool to the function pool, use the VGET service with the PROFILE keyword.

If the variable is a system variable, whose name begins with the letter Z, the value in the shared pool is updated. If the variable was created with VDEFINE, with a pointer in the function pool, the value in user storage is updated. This is illustrated in Figure 13.

```
CALL DGILINK('VGET ','(ABC GHI XXX)','PROFILE ');
```

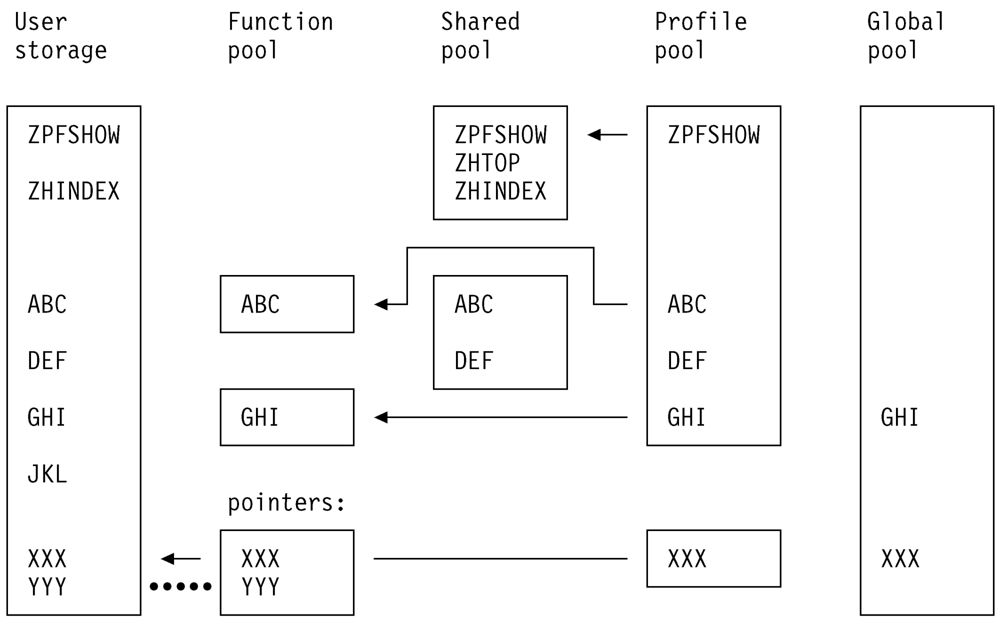


Figure 13. VGET service with PROFILE keyword (non-REXX)

To copy the value of a variable from the global pool to the function pool, use the VGET service with the GLOBAL keyword. If the variable was created with VDEFINE, with a pointer in the function pool, the value in user storage is updated. This is illustrated in Figure 14.

```
CALL DGILINK('VGET ','(GHI XXX)','GLOBAL ');
```

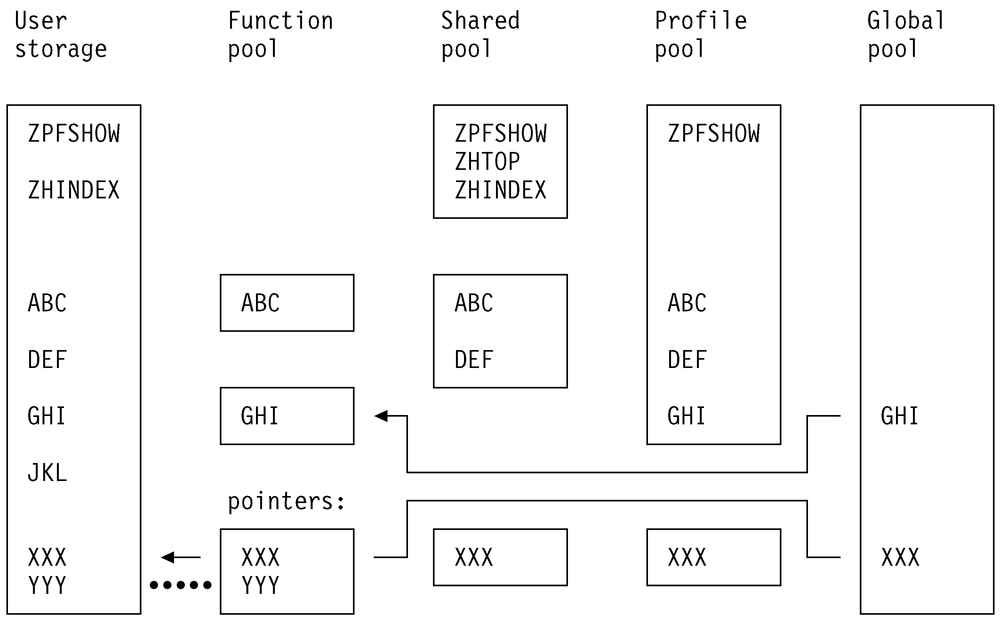


Figure 14. VGET service with GLOBAL keyword (non-REXX)

If you specify VGET with the ASIS keyword or with no keyword, the value in the shared pool (if any) is copied. If the variable does not exist in the shared pool, the value in the profile pool is copied.

VPUT service

To copy the value of a variable from the function pool to the shared pool, use the VPUT service with the SHARED keyword. If the variable was created with VDEFINE, with a pointer in the function pool, the value in user storage is copied to the shared pool. This is illustrated in Figure 15.

```
CALL DGILINK('VPUT ','(ABC XXX)','SHARED ');
```

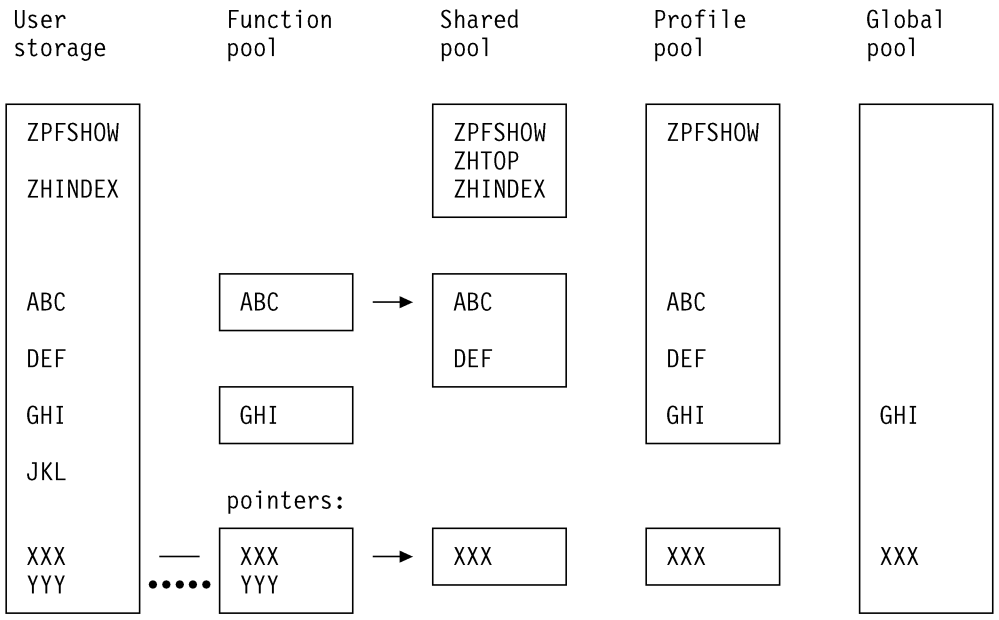


Figure 15. VPUT service with SHARED keyword (non-REXX)

To copy the value of a variable from the function pool to the profile pool, use the VPUT service with the PROFILE keyword.

If the variable was created with VDEFINE, with a pointer in the function pool, the value in user storage is copied to the profile pool. If the variable is a system variable (whose name begins with the letter Z), the value is copied from the shared pool. This is illustrated in Figure 16 on page 22.

```
CALL DGILINK('VPUT ','(ZPFSHOW ABC GHI XXX)','PROFILE ');
```

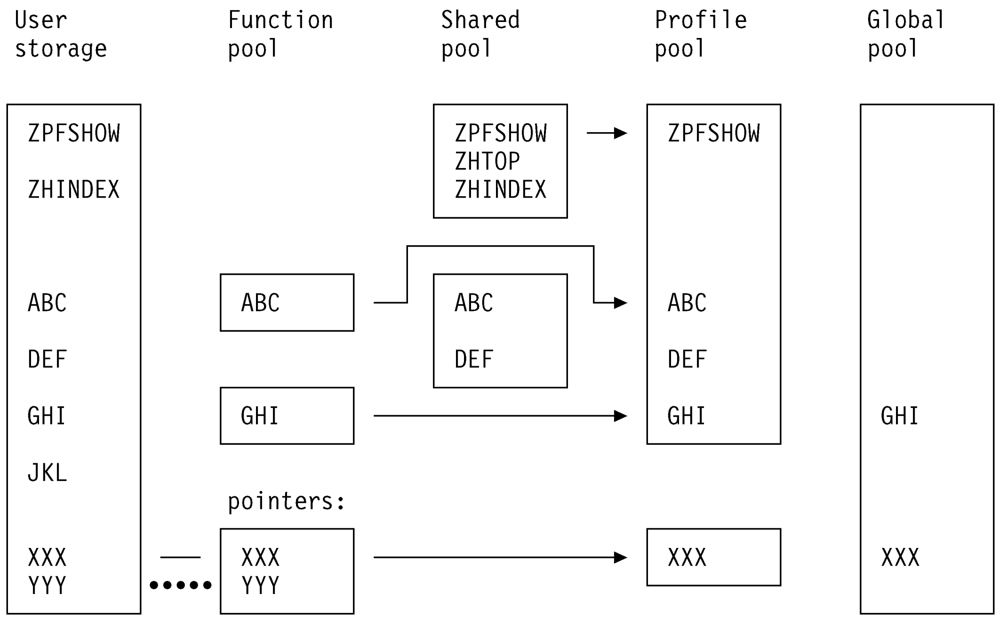


Figure 16. VPUT service with PROFILE keyword (non-REXX)

To copy the value of a variable from the function pool to the global pool, use the VPUT service with the GLOBAL keyword. If the variable was created with VDEFINE, with a pointer in the function pool, the value in user storage is copied to the global pool. This is illustrated in Figure 17.

```
CALL DGILINK('VPUT ','(GHI XXX)','GLOBAL ');
```

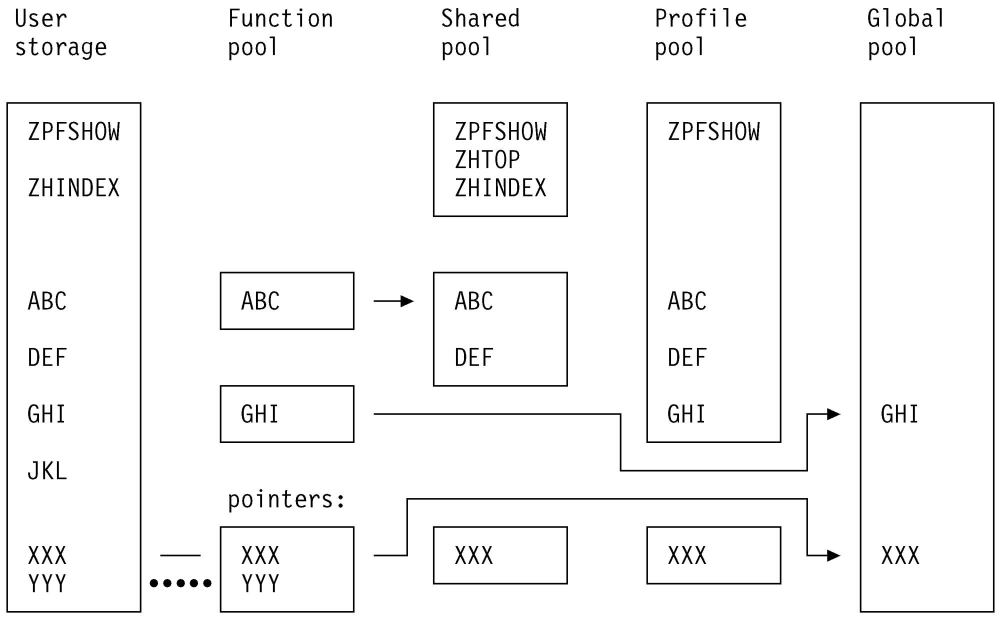


Figure 17. VPUT service with GLOBAL keyword (non-REXX)

If you specify VPUT with the ASIS keyword or with no keyword, the value in the shared pool (if any) is updated. If the variable does not already exist in the shared pool, the value in the profile pool (if any) is updated. If the variable exists in neither the shared pool nor the SDF2/DM profile, a new variable is created in the shared pool.

VERASE service

To erase a variable from the shared pool, use the VERASE service with the SHARED keyword. Any value in the function pool is also erased. This is illustrated in Figure 18.

```
CALL DGILINK('VERASE ','(ABC DEF)','SHARED ');
```

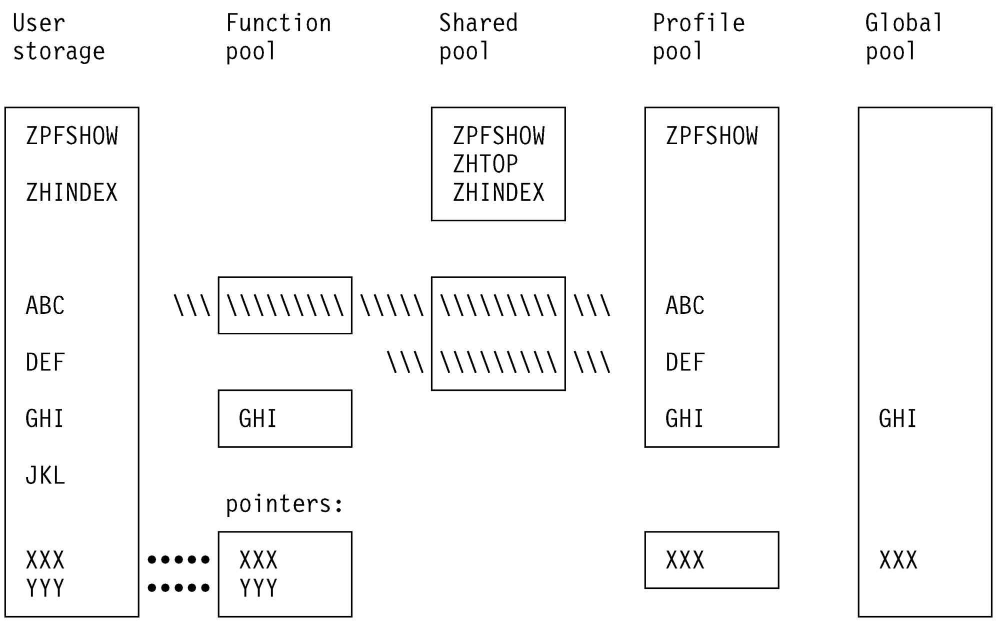


Figure 18. VERASE service with SHARED keyword (non-REXX). Erasure of the variable is indicated by the \ characters.

To erase a variable from the profile pool, use the VERASE service with the PROFILE keyword. If the variable exists also in the function pool, it is erased from there, too. This is illustrated in Figure 19.

```
CALL DGILINK('VERASE ','(ABC XXX)','PROFILE ');
```

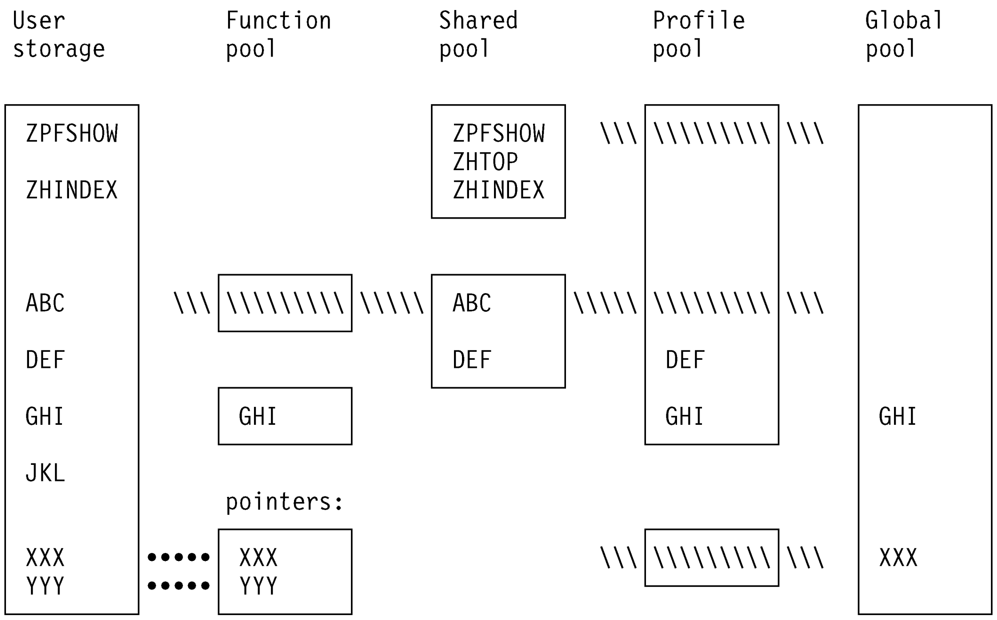


Figure 19. VERASE service with PROFILE keyword (non-REXX). Erasure of the variable is indicated by the \ characters.

To erase a variable from both the shared pool and the profile pool, use the VERASE service with the BOTH keyword.

If you specify VERASE with the ASIS keyword or with no keyword, the value in the shared pool (if any) is erased. If the variable does not exist in the shared pool, the variable is erased from the profile pool instead.

VDEFINE service

To create an SDF2/DM variable with a pointer to the program variable in the user storage, use the VDEFINE service. This is illustrated in Figure 20.

```
CALL DGILINK('VDEFINE ','XXX ',...);
```

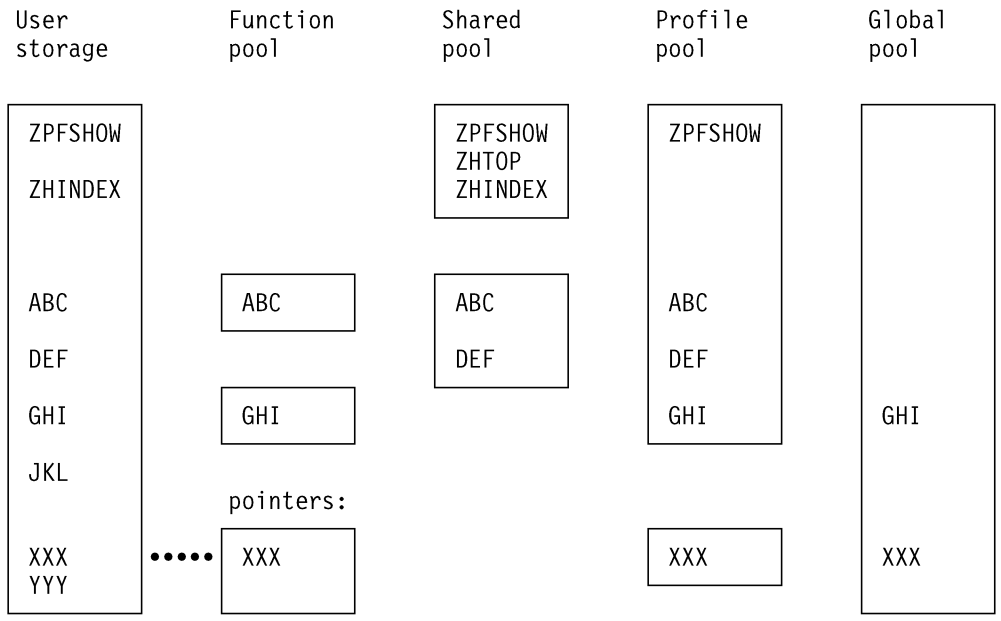


Figure 20. VDEFINE service (non-REXX)

If the variable already exists in the function pool, or if you have already defined the variable at a different location in user storage by using VDEFINE, you create a new version of the variable. This new version temporarily overrides any other versions, until you erase it with the VDELETE service.

A variable that is created with VDEFINE can be translated to a bit string, fixed binary string, or hexadecimal string. The translation occurs automatically when the variable is stored by an SDF2/DM service. A translation back to character string format occurs automatically when the variable is accessed.

When a defined variable is stored, either of two errors can occur:

Truncation	If the current length of the variable is greater than the defined length within the module, the remaining data is lost.
Translation	If the variable is defined as something other than a character string, and the external representation has invalid characters, the contents of the defined variable are lost.

In either case, the SDF2/DM service issues a return code of 16.

VDELETE service

To erase a variable that was created with VDEFINE, use the VDELETE service. This is illustrated in Figure 21.

```
   CALL DGILINK('VDELETE ','XXX ',...);
```

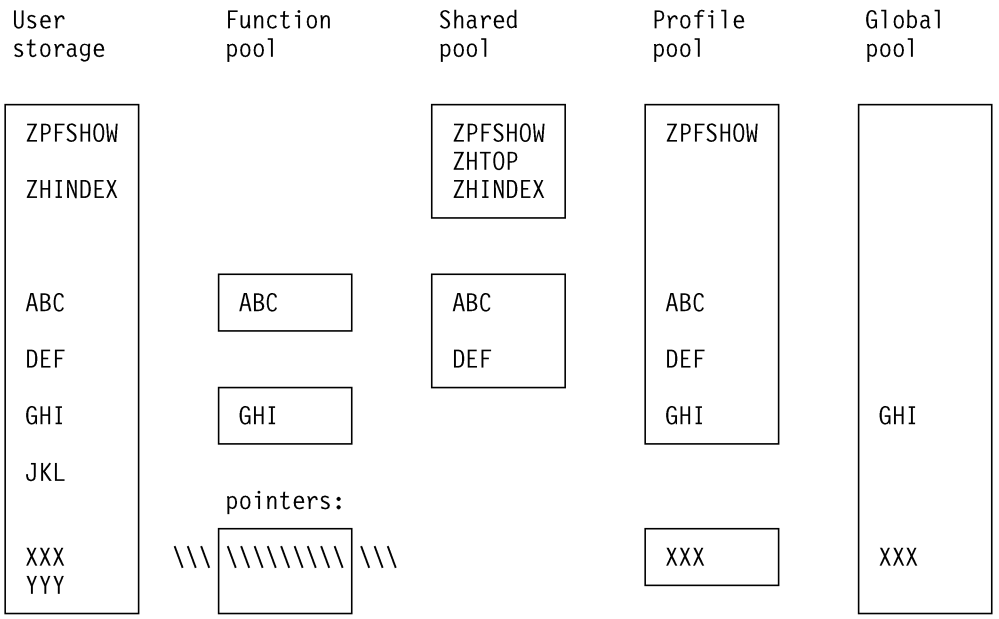


Figure 21. VDELETE service (non-REXX). Erasure of the variable is indicated by the \ characters.

If several versions of the variable exist, VDELETE deletes only the most recent version. The previous version is then used.

For example, in a subroutine you can create a variable named XXX using VDEFINE and later delete it using VDELETE. You do not need to worry about whether XXX already exists. The previous XXX value, if any, will be available again when you issue the VDELETE call at the end of the subroutine.

VDELETE affects only the function pool. The actual value of the variable, in user storage, is not affected.

VRESET service

To erase the function pool, use the VRESET service. Pointers to variables that were created with VDEFINE are also erased. This is illustrated in Figure 22.

```
CALL DGILINK('VRESET ');
```

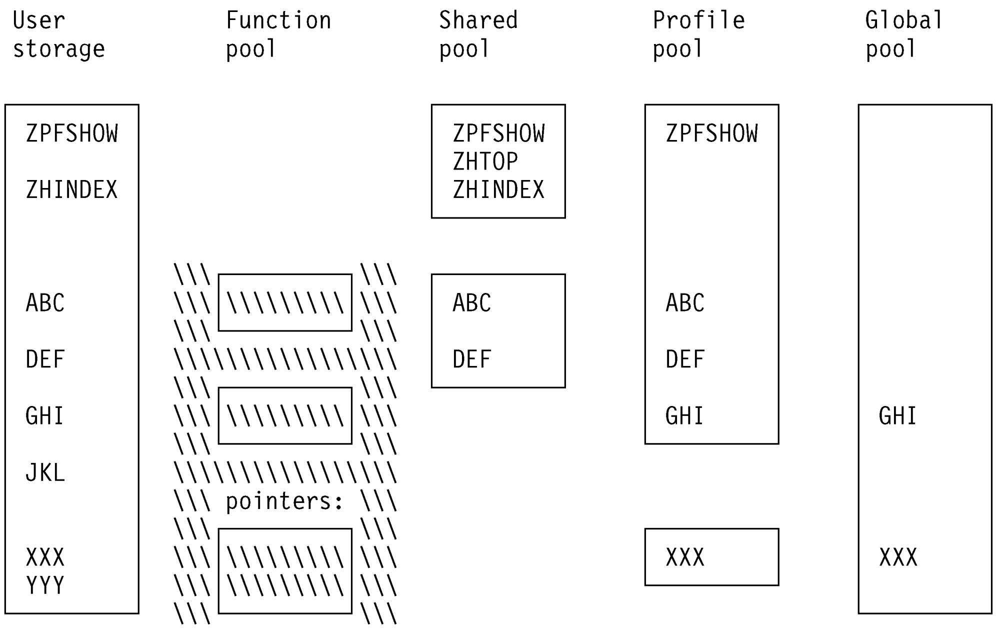


Figure 22. VRESET service (non-REXX). Erasure of the variable is indicated by the \ characters.

Using the table services

Table services let you use and maintain sets of dialog variables. A table is a two-dimensional array of information in which each column corresponds to a dialog variable, and each row contains a set of values for those variables.

Contents for a table are shown in Figure 23.

EMPSER	LNAME	FNAME	I	PHA	PHNUM
598304	Gaicki	Richard	P	301	840-1224
172397	Puchas	Leopold	A	301	547-8465
813058	Heihs	Michael	L	202	338-9557
395733	Schuster	Günter	Q	202	477-1776
502774	Pesendorfer	Karl	A	914	555-4156

Figure 23. Five rows in table TAB1

In this example, the variables that define the columns are as follows:

EMPSER	Employee serial number
LNAME	Last name
FNAME	First name
I	Middle initial
PHA	Home phone: Area code
PHNUM	Home phone: Local number

Where tables reside

A table can be either temporary or permanent. A temporary table exists only in virtual storage. It cannot be written to disk storage.

Permanent tables are maintained in one or more table libraries. A permanent table, while created in virtual storage, can be saved on direct access storage. It can be opened for update or for read-only access, at which time the entire table is read into virtual storage. When a table is being updated in virtual storage, the copy of the table on direct access storage cannot be accessed until the update is complete.

For both temporary and permanent tables, rows are accessed and updated from the in-storage copy. A permanent table that has been accessed as read-only can be modified in virtual storage, but cannot be written back to disk storage.

When a permanent table is opened for processing, it is read from a table input library. A table to be saved can be written to a table output library that is different from the input library. The input and output libraries should be the same if the updated version of the table is to be reopened for further processing by the same dialog.

Services that affect an entire table

The following services operate on an entire table:

TBCLOSE	Closes a table and saves a permanent copy if the table was opened.
TBCREATE	Creates a new table and opens it for processing.
TBEND	Closes a table without saving it.
TBERASE	Deletes a permanent table from the table output file.
TBOPEN	Opens an existing permanent table for processing.
TBQUERY	Obtains information about a table.
TBSAVE	Saves a permanent copy of a table without closing it.
TBSORT	Sorts a table.
TBSTATS	Provides access to statistics for a table.

Temporary tables are created by the TBCREATE service (nowrite mode) and deleted by either the TBEND or TBCLOSE service. A new permanent table is created in virtual storage by the TBCREATE service (write mode). The table does not become permanent until it is stored on direct access storage by either the TBSAVE or TBCLOSE service.

An existing permanent table is opened and read into virtual storage by the TBOPEN service. If the table is to be updated (write mode), the new copy is saved by either the TBSAVE or TBCLOSE service. If it is not to be updated (nowrite mode), the virtual storage copy is deleted by either the TBEND or TBCLOSE service.

Services that affect table rows

The following services operate on a row of the table:

TBADD	Adds a new row to the table.
TBBOTTOM	Sets the CRP to the last row and retrieves the row.
TBDELETE	Deletes a row from the table.
TBEXIST	Tests for the existence of a row (by key).
TBGET	Retrieves a row from the table.

TBMOD	Updates an existing row in the table or adds a new row to the table.
TBPUT	Updates a row in the table if it exists and if the keys match.
TBSARG	Establishes a search argument for use with TBSCAN. It can also be used in conjunction with TBDISPL.
TBSCAN	Searches a table for a row that matches a list of argument variables, and retrieves the row.
TBSKIP	Moves the CRP forward or back by a specified number of rows, and then retrieves the row at which the CRP is positioned.
TBTOP	Sets the CRP to TOP (zero), ahead of the first row.
TBVCLEAR	Sets to null any dialog variables that correspond to variables in the table.

Accessing data

You specify the variable names that define table columns when the table is created. Specify each variable as either a key field or a name (non-key) field. You can specify one or more columns (variable names) as keys for accessing the table. For the table shown in Figure 23 on page 27, EMPSER might be defined as the key variable. Alternatively, EMPSER and LNAME might both be defined as keys, in which case, a row would be found only if both EMPSER and LNAME match the current values of those variables. A table can also be accessed by one or more *argument* variables, which need not be key variables. You can define the variables that constitute the search argument dynamically by using the TBSARG and TBSCAN services.

In addition, a table can be accessed by use of the current row pointer (CRP). The table can be scanned by moving the CRP forward or backward. A row can be retrieved each time the CRP is moved. When a table is opened, the CRP is automatically positioned at the top of the table (TOP), ahead of the first row. Table services, such as TBTOP, TBBOTTOM, and TBSKIP are available for positioning the CRP.

When a row is retrieved from a table, the contents of the row are stored in the corresponding dialog variables. When a row is updated or added, the contents of the dialog variables are saved in that row.

When a row is stored, a list of *extension* variables can be specified by name. These extension variables, and their values, are added to the row. Thus, variables that were not specified when the table was created can be stored in the row. A list of extension variable names for a row can be obtained when the row is read. If the list of extension variables is not respecified when the row is rewritten, the extensions are deleted.

SDF2/DM table services treat blank data and NULL (zero-length) data as equal. For example, the following VDEFINE requests are executed:

```
"DGILINK('VDEFINE ','(V1)',VAL1,'CHAR ',L8,' NOBSCAN ')"
"DGILINK('VDEFINE ','(V2)',VAL2,'CHAR ',L8)"
```

sip If L8 = 8, VAL1 = 'ABCDƀƀƀƀ' and VAL2 = 'ABCDƀƀƀƀ', where ƀ is a blank character, V1 will have a length of 8 and a value of 'ABCDƀƀƀƀ', and V2 will have a length of 4 and a value of 'ABCD'. To SDF2/DM, V1 and V2 will be equal, because when SDF2/DM compares two values, it first pads the shorter value with blanks so that the lengths are equal, then compares the values.

If the same VDEFINE requests are executed with VAL1 = 'ƀƀƀƀƀƀƀƀ' (8 blanks) and VAL2 = '', V1 will have a length of 8 and a value of 'ƀƀƀƀƀƀƀƀ' (8 blanks), and V2 will have a length of 0 (NULL value). To SDF2/DM, V1 is equal to V2, since SDF2/DM pads V2 with 8 blanks before comparing V2 with V1.

Example: Create and update a simple table

The following series of commands demonstrates the use of table services:

1. Create a permanent table, named DALPHA, consisting of dialog variables AA, BB, and CC. AA is the key field. BB and CC are name fields.

   ```
   DGIEXEC TBCREATE DALPHA KEYS(AA) NAMES(BB CC) WRITE
   ```

   The DALPHA table might consist of the following entries:

AA	BB	CC
Zamecnik Johann	W590	Jones Beach
Schwarzmüller Harald	Y200	Bar Harbour

   Table services adds a row to table DALPHA immediately following the row added by the previous TBADD. Following the TBADD, the current row pointer (CRP) is positioned at the newly added row. Before a row is added by the TBADD service, table service scans the table to determine whether the KEY field of the new row to be added duplicates the KEY field of an existing row. If it does, the TBADD is not performed, and a message is displayed.

2. Save table DALPHA for later use by writing it to the table output library.

   ```
   DGIEXEC TBCLOSE DALPHA
   ```

   The table DALPHA is written from virtual storage to the file specified by DGITABL.

Using the file-tailoring services

File-tailoring services read skeleton files and write tailored output that can be used to drive other functions. Frequently, file tailoring is used to generate job files for batch execution.

The file-tailoring services, listed in the order in which they are normally invoked, are:

FTOPEN	Prepares the file-tailoring process and specifies whether the temporary file is to be used for output
FTINCL	Specifies the skeleton to be used and starts the tailoring process
FTCLOSE	Ends the file-tailoring process
FTERASE	Erases an output file created by file tailoring

The file-tailoring output is written to a VSE sublibrary. If you do not specify the member and library name explicitly, the output is written to the sublibrary DGIWORK.DGI, and given a generated name that is available in system variable ZTEMPF.

Skeleton files

Each skeleton file is read record-by-record. Each record is scanned to find any dialog variable names, which are names preceded by an ampersand. When a variable name is found, its current value is substituted from a variable pool.

Skeleton file records can also contain statements that control processing. These statements provide the ability to do the following:

- Set dialog variables
- Imbed other skeleton files
- Conditionally include records
- Iteratively process records in which variables from each row of a table are substituted

When iteratively processing records, file-tailoring services read each row from a specified table. If the table was already open prior to processing the skeleton, it remains open with the current row pointer (CRP) set to TOP (zero). If the table was not already open, file tailoring opens it automatically and closes it on completion of processing.

Example of a skeleton file

A sample skeleton file is shown in Figure 24. The tailored output could be submitted to the background for processing.

```
)SKELETON
* $$ JOB JNM=ASMBMS,CLASS=5,DISP=D,USER=SCHWARZM
// JOB ASMAOPTV ASSEMBLE DEFAULT OPTIONS
// SETPARM CLOSE='NO'
// ON $ABEND  GOTO RESET
// ON $CANCEL GOTO RESET
// ON $RC > 4 GOTO RESET
/. C -----------------------------------------------------
/. C The following group of SETC statements will
/. C generally require your customization
/. C -----------------------------------------------------
// SETPARM PUNVOL='SYSWK2'  * SYSPCH file specifications
// SETPARM START='0227'     * SYSPCH extent begin
// SETPARM LENGTH='0020'    * SYSPCH extent length
// SETPARM LIB='BMSLIB'     * OBJ deck target library
// SETPARM SUB='OBJLIB'     * OBJ deck target sublibrary
// SETPARM LIBVOL='SYSWK2'  * OBJ library VOLSER
// SETPARM LIBID='BMS.OBJ.LIB' * OBJ library file ID
/. C -----------------------------------------------------
/. C End of customization section
/. C -----------------------------------------------------
*
*  ASSEMBLE &BMSMAP CICS/BMS OBJECT
*
)CM
)CM    generate search chain for my job
)CM
)SEL &SEARCHL1 = &Z
// LIBDEF *,SEARCH=(PRD2.PROD)
)ENDSEL
)SEL &SEARCHL1 ¬= &Z
)SEL &SEARCHL2 = &Z
// LIBDEF *,SEARCH=(PRD2.PROD,&SEARCHL1)
)ENDSEL
)SEL &SEARCHL2 ¬= &Z
// LIBDEF *,SEARCH=(PRD2.PROD,&SEARCHL1,&SEARCHL2)
)ENDSEL
)ENDSEL
// DLBL IJSYSPH,'ASSEMBLE.OUTPUT',0
// EXTENT SYSPCH,&&PUNVOL,1,0,&&START,&&LENGTH
ASSGN SYSPCH,DISK,VOL=&&PUNVOL,SHR
// SETPARM CLOSE='PUN'
// OPTION DECK
// EXEC ASMA90
         PUNCH 'CATALOG &BMSMAP..OBJ REPLACE=YES'
         PRINT ON,NOGEN
)CM
)CM    include BMS source for assembly
)CM
)IM BMSSRC
/*
CLOSE SYSPCH,PUNCH
// SETPARM CLOSE='NO'
// DLBL &&LIB,'&&LIBID',0
// EXTENT ,&&LIBVOL
// DLBL IJSYSIN,'ASSEMBLE.OUTPUT',0
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=&&PUNVOL,SHR
// SETPARM CLOSE='IPT'
// EXEC LIBR,PARM='AC S=&&LIB..&&SUB'
/*
/. RESET ASSIGNMENTS
// IF CLOSE = 'IPT' THEN
CLOSE SYSIPT,SYSRDR
// IF CLOSE = 'PUN' THEN
CLOSE SYSPCH,PUNCH
/*
/&&
* $$ EOJ
)END
```

Figure 24. Sample skeleton file

The sample skeleton refers to several dialog variables, including BMSMAP, SEARCHL1, and SEARCHL2, which are highlighted in Figure 24 on page 32. It also illustrates the use of)SEL and)ENDSEL select statements to conditionally include records. The first part of the example has nested SELECT statements to include concatenated search libraries if the library names have been specified by the user (that is, if variables SEARCHL1 and SEARCHL2 are not equal to the null variable Z).

In the second part of the example, an imbed statement,)IM, is used to bring in a separate skeleton for the BMS source input to be assembled.

Example of using file-tailoring services

The following example illustrates file-tailoring services. For this example, assume the following:

- LABLSKEL is a member in the file-tailoring library, containing:

Skeleton LABLSKEL

```
)SKELETON
)DOT DALPHA
     NAME:   &AA
APARTMENT:   &BB
     CITY:   &CC
     YEAR:   &ZSTDYEAR
)ENDOT
)END
```

- ZSTDYEAR is the name of an SDF2/DM system variable that contains the current year.
- DALPHA is a table:

AA	BB	CC
Zamecnik Johann	W590	Jones Beach
Schwarzmüller Harald	Y200	Bar Harbour

This example creates a name-and-address list. The file-tailoring service requests are:

- `DGIEXEC FTOPEN`

  FTOPEN opens both the file-tailoring skeleton and file-tailoring output files. These files must be defined to SDF2/DM before starting the SDF2/DM session.

- `DGIEXEC FTINCL LABLSKEL`

  FTINCL performs the file-tailoring process by using the file-tailoring skeleton named LABLSKEL. LABLSKEL contains the file-tailoring controls,)DOT and)ENDDOT, which specify the use of table DALPHA.

- `DGIEXEC FTCLOSE NAME(LABLOUT)`

  Write the resulting file-tailoring output to a member named LABLOUT SKELETON.

At the conclusion of processing the previous commands, file-tailoring output file LABOUT SKELETON contains:

```
     NAME:   Zamecnik Johann
APARTMENT:   W590
     CITY:   Jones Beach
     YEAR:   1984
     NAME:   Schwarzmüller Harald
APARTMENT:   Y200
     CITY:   Bar Harbour
     YEAR:   1984
```

Defining menus

A menu is a panel on which a user can see which options are available, and can select one. Menus are often designed as follows:

- Various commands and other options are listed.
- To the left of each choice is a unique 1-digit or 1-letter identifier.
- To select a choice, the user enters the identifier character.

A menu, also called a selection panel, is a special type of panel. Menus are processed by the SELECT service. The sections that can be used in a menu definition are the same as those that can be used in other panel definitions. However, a menu requires a processing section in addition to the body section. The processing section must set the variable ZSEL.

The processing section of a menu has the following general format.

```
)PROC
 &ZSEL = TRANS( TRUNC(&ZCMD,'.')
                value, 'string'
                value, 'string'
                     .
                     .
                value, 'string'
                  ' ', ' '
                    *, '?' )
```

The ZCMD variable is truncated prior to translation, so users can bypass one or more intermediate menus. For example, 1.2 means primary option 1, suboption 2. When the SELECT service discovers that .TRAIL is not blank (when it contains a period) it causes the next lower-level menu to be selected with an initial option of the value following the period in .TRAIL. As long as the initial option is not blank, the lower-level menu is processed in the normal fashion but is not displayed to the user.

value is an option that can be entered on the menu.
string contains selection keywords indicating the action to occur.

The selection keywords are:

PANEL(*panel-name***)**	[**OPT(***option***)**] [**NEWAPPL(***applid***)**]
	[**POPUP**] [**POPLOC(***field-name***)**] [**ROW(***row***)**] [**COLUMN(***column***)**]
PGM(*program-name***)**	[**PARM(***parameters***)**] [**NEWAPPL(***applid***)**]
CMD(*command***)**	[**NEWAPPL(***applid***)**]
[**HELP**]	
[**EXIT**]	

Except for HELP and EXIT, the keywords are the same as those that can be specified for the SELECT service. For information on the keywords, see "SELECT — Select a panel or function" on page 155.

HELP
The HELP keyword calls the help manager to display the specified help panel. You can use it to display your tutorial panels.

EXIT
The EXIT keyword applies only to primary menu panels. It terminates the SELECT service with a return code of 4.

Blank options
If you translate the option to a blank:

```
' ',' '
```

SDF2/DM displays the message `Enter selection`.

Invalid options
If you translate the option to a question mark:

```
*,'?'
```

SDF2/DM displays the message `Invalid selection`.

Providing help

This section describes how to make help available to the users of your application prototype.

A help panel is coded like any other panel. If you want the help panel to appear in a pop-up window, specify the dimensions of the window in the WINDOW keyword of the)BODY header statement. If you want a title to appear on the border of the help pop-up window, set ZHELPTTL equal to the title in the)INIT section, and do not put the title in the)BODY section.

Within a panel definition or within your application prototype, specify which help panel is to be displayed in different circumstances. The way that you do this depends on the type of help involved.

Coding message help

For each message, you can specify the help panel that corresponds to the message on the .HELP keyword in the message definition. If you do not specify a .HELP keyword, extended help is used.

For example:

```
EMPX210   'INVALID TYPE OF CHANGE' .HELP=PERSO33   .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'

EMPX213    'ENTER FIRST NAME'       .HELP=PERSO34  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX216    'AREA CODE INVALID'      .ALARM=YES
'AREA CODE &PHA IS NOT DEFINED. PLEASE CHECK THE PHONE BOOK.'
```

If the user requests help for message EMPX210, panel PERSO33 appears. If the user requests help for message EMPX213, panel PERSO34 appears. If the user requests help for message EMPX216, extended help for the panel on which message EMPX216 was displayed appears.

Coding field help

For field help:

1. Name the field.
2. In the)HELP section of the panel definition, either identify the help panel that is to be displayed or specify that the help command be returned to the application prototype.

Each input or output field automatically has the name of the variable associated with the field. For an action-bar choice or pull-down choice, you can specify a name on the)ABC or PDC statement. For a reference phrase, you follow the instructions in “Coding reference phrase help” on page 37.

For example:

```
)ATTR
 # TYPE(NT)
)BODY
  ⋮
%COMMAND ===> _ZCMD                                             %
  ⋮
)HELP
HELP FIELD(ZCMD) PANEL(HLPZCMD)
)END
```

Figure 25. Field help example

Coding reference phrase help

Reference phrase help is a special case of field help. Within a panel, you can identify reference phrases and assign to each a specific help panel. When the panel is displayed, each reference phrase is highlighted. The user can move from one reference phrase to the next with the tab keys. To support tabbing, each reference phrase is considered an input-capable field, which also means that the reference phrase is refreshed every time the screen is redisplayed.

To define reference phrase help, do the following:

- In the)ATTR section, assign an attribute character with TYPE(RP)
- In the)BODY and)AREA sections, mark the beginning of each reference phrase with the RP attribute character and mark the end of each reference phrase with a text attribute character.
- The reference phrase field can contain a variable name. In this case the text is substituted when the panel is displayed.
- In the)HELP section, assign a help panel to each reference phrase. The field name for the reference phrase is ZRP*xxyyy*, where:

  xx Is 00 if the reference phrase appears in the)BODY section, or a number from 01 to 99 if it appears in an)AREA section. The area sections are numbered, based on where they appear within the panel body. Numbering starts at the top left-hand corner of the panel and continues through each line of the panel from left to right.

  yyy Is the number of the reference phrase within its)BODY or)AREA section, from 001 to 999. Numbering starts at the top left-hand corner of the)BODY or)AREA section and continues through each line of the section from left to right.

If a reference phrase continues over more than one line, you can mark the part of the phrase that appears on each line as a separate reference phrase. If you then assign the same help panel to each of these reference phrases, they function logically as a single reference phrase.

You can use the .ZRP control variable to assign meaningful variable names to reference phrases.

For example:

```
)PANEL
)ATTR
  #  TYPE(RP)
)BODY
+This is sample text.  This is a #Reference Phrase+.
+This is an example of a #Reference Phrase being
 physically continued to the next line.+
+This is an example of a #Reference Phrase being+
#logically continued to the next line.+
+
)HELP
 FIELD(ZRP00001)  PANEL(BODY0001)
 FIELD(ZRP00002)  PANEL(BODY0002)
 FIELD(ZRP00003)  PANEL(BODY0003)
 FIELD(ZRP00004)  PANEL(BODY0003)
)END
```

Figure 26. Reference phrase help example

The body section contains four reference phrases. The last two phrases share the same help panel, and thus function as a single reference phrase.

Coding extended help

For extended help, assign a value to the control variable .HELP. Typically, you do this in the)INIT section of a panel definition.

For example:

```
)INIT
  .HELP=HP001
```

The panel HP001 is the extended help panel for the panel that you are defining.

Coding keys help

For keys help, either specify a panel name on the)KEYLIST statement or assign a value to the system variable ZKEYHELP.

For example:

```
)INIT
  &ZKEYHELP=KH025
```

specifies that help panel KH025 is to be displayed when KEYSHELP is requested. You can achieve the same effect with the following statement:

```
)KEYLIST HELP(KH025)
```

Coding the help index

The help index is an alphabetical list of help topics. Each topic points to a panel that contains help information. You can point to extended help panels, field help panels, or panels that are not referred to elsewhere in the dialog. You can point to the same panel from more than one topic.

Whenever a user requests the index (by issuing the **index** command) from within help, the help index panel is displayed.

At the beginning of your application prototype, set the value of ZHINDEX to the name of your help index panel.

Coding the tutorial

The tutorial is a set of panels that describe how to use your application. It can consist of extended help panels or panels that are not referred to elsewhere in the dialog.

At the beginning of your application prototype, set the value of ZHTOP to the name of the main menu of your tutorial. Whenever a user requests the table of contests (by issuing the **toc** command) from within help, this tutorial menu is displayed.

Chapter 2. Panel definition sections

This chapter describes the sections of a panel definition. Most sections are optional, but if they are included they must be in the sequence shown here. Figure 27 shows the sections of a panel definition.

Section	Requirement	Purpose
)PANEL	Optional	Identifies the panel as a CUA panel
)ATTR	Optional	Identifies attribute characters
)ABC	Optional	Defines an action-bar choice for a panel and its associated pull-down choices
)ABCINIT	Optional	Contains processing instructions to be performed when a user selects an action-bar choice
)ABCPROC	Optional	Contains initialization instructions to be performed when a user finishes interacting with a pull-down
)BODY	Required	Specifies the format of the panel as the user sees it
)AREA	Optional	Defines scrollable areas on a panel
)LIST	Optional	Defines scrollable lists on a panel
)INIT	Optional	Specifies the initial processing to be performed when a panel is displayed
)REINIT	Optional	Specifies the processing to be performed when a panel is redisplayed
)PROC	Optional	Specifies the processing to be performed when a user finishes interacting with a panel
)HELP	Optional	Specifies which help panel is to be displayed when a user requests help for a panel element
)KEYLIST	Optional	Specifies what happens when a user presses a function key while a panel is being displayed
)END	Required	Signals the end of a panel definition

Figure 27. Panel definition sections

In SDF II application prototypes, the generated panels use the syntax described here.

The)PANEL section

The)PANEL section is optional. It consists solely of the)PANEL statement, which identifies the panel as a CUA panel. On a CUA panel, the EXIT or CANCEL command issues a return code of 8 for the DISPLAY request.

Syntax

)PANEL [**KEYLIST (***keylist-name***)**]

keylist-name
The name of the keylist to be used for the panel. You can specify a variable name preceded by an ampersand. This lets you dynamically select a keylist at run time.

You can either include a keylist in the)KEYLIST section, or make the keylist a separate file and refer to the file on the)PANEL statement. The named keylist does not need to exist when the panel definition is created, but must exist at run time.

For more information about keylists, see Chapter 5, “Keylists” on page 107.

The)ATTR section

The)ATTR (attribute) section of a panel is optional. It identifies the special characters that you will use as attribute characters (start- and end-of-field characters) within the)BODY section. When the panel is displayed, attribute characters are replaced with the appropriate hardware attribute bytes, which appear on the screen as blanks. If you use only the default attribute characters, you do not need an)ATTR section.

You can specify attribute characters in the following places:

- The DEFAULT keyword on the)ATTR or)BODY statement
- Attribute statements within the)ATTR section

The)ATTR section begins with the)ATTR header statement and ends with the)ABC or)BODY statement.

Note: You can identify some attribute characters on the DEFAULT keyword of the)ATTR or)BODY statement. If you specify a DEFAULT keyword on both statements, the)BODY statement takes precedence.

Syntax

)ATTR [**DEFAULT(***def1def2def3***)**]

where:

def1	Represents high-intensity text fields
def2	Represents low-intensity text fields
def3	Represents high-intensity input fields

The value inside the parentheses must consist of exactly three characters, which must not be enclosed in single quotes or be separated by commas or blanks.

The default attribute characters are:

% (percent sign)	Text (protected) field, high intensity
+ (plus sign)	Text (protected) field, low intensity
_ (underscore)	Input (unprotected) field, high intensity

The values that you specify in the DEFAULT keyword override these defaults. For example, if you specify:

```
)ATTR DEFAULT($¢_)
```

the $ character will represent high-intensity text fields, the ¢ character will represent low-intensity text fields, and the _ character (the default) will represent high-intensity input fields. The values % and + can be assigned to other attributes or can be used within the text that appears on the panel.

Attribute statements

If you want to use additional attribute characters (beyond the three attribute characters specified on the DEFAULT keyword), you use attribute statements within the)ATTR section.

Each attribute statement consists of a unique attribute character (either the character itself or the 2-digit hexadecimal number that corresponds to the character) followed by one or more keywords. Keywords can be continued onto subsequent lines.

The following example illustrates attribute statements, some of which use the hexadecimal representation of the attribute code:

```
)ATTR
  + TYPE(NT)                        /* normal text              4E */
  % TYPE(ET)                        /* emphasized text          6C */
  " TYPE(DT)                        /* descriptive text         7F */
  _ TYPE(NEF) CAPS(ON)              /* normal entry f.w. caps on  6D */
  ¬ TYPE(NEF)                       /* normal entry field       5F */
  \ TYPE(PIN)                       /* panel instructions       E0 */
  $ TYPE(FP)                        /* field prompt             5B */
  } TYPE(CH)                        /* column heading (r. brace)  D0 */
  @ TYPE(SF)                        /* selection field          7C */
 23 TYPE(EE) CAPS(ON)               /* error emphasis           23 */
 20 TYPE(AB)                        /* action bar               20 */
  ¢ TYPE(VOI)                       /* variable output inf.     4A */
 33 TYPE(OUTPUT)  CUATYPE(SI)       /*                          33 */
```

Generally, you should choose special (non-alphanumeric) characters for attribute characters, so that they will not conflict with the panel text. An ampersand (&), blank (hexadecimal 40), shift-out (hexadecimal 0E), shift-in (hexadecimal 0F), or null (hexadecimal 00) cannot be used as an attribute character.

You can specify a maximum of 127 attribute characters. This limit includes the three default characters, and attribute overrides.

In the)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC sections of the panel definition, you can use the .ATTR and .ATTRCHAR control variables to change the values of some keywords. For more information, see Chapter 4, “Control variables” on page 100.

Syntax of the attribute statement

There are various possible syntaxes for the attribute statement, depending on what the attribute character refers to. In each syntax, keywords can be specified in any order.

If the attribute character is used for a scrollable area, the following syntax is used:

Syntax

char **AREA(SCRL)** [**EXTEND(ON**|**OFF)**]

If the attribute character is used for a scrollable list, the following syntax is used:

Syntax

```
char AREA(LIST) [EXTEND(ON|OFF)]
```

If the attribute character is used for a dynamic area, the following syntax is used:

Syntax

```
char AREA(DYNAMIC) [EXTEND(ON|OFF)]
                   [SCROLL(ON|OFF)]
                   [USERMOD(usermod-code)]
                   [DATAMOD(datamod-code)]
```

In all other cases, the following syntax is used:

Syntax

```
char TYPE(value) [COLOR(value)]
                 [INTENS(HIGH|LOW|NON)]
                 [HILITE(USCORE|BLINK|REVERSE)]
                 [CUATYPE(value)]
                 [CAPS(ON|OFF|IN|OUT)]
                 [JUST(LEFT|RIGHT|ASIS)]
                 [PAD(NULLS|char)]
                 [PADC(NULLS|char)]
                 [SKIP(ON|OFF)]
                 [NUMERIC(ON|OFF)]
                 [FORMAT(EBCDIC|DBCS|MIX)]
```

The remainder of this section does not discuss the AREA(SCRL), AREA(LIST), and AREA(DYNAMIC) syntaxes. For information about scrollable areas, see "The)AREA section" on page 69. For information about scrollable lists, see "The)LIST section" on page 76. For information about dynamic areas, see "Dynamic areas" on page 62.

The TYPE keyword

Figure 28, Figure 29, and Figure 30 on page 44 show the values that the TYPE keyword can have. The attribute types are divided into the following categories:

Basic Used for input fields, output fields, and static text.

Dynamic area Can be used within a dynamic area. Within a dynamic area, DATAIN, DATAOUT, and DATAOUTGE replace the basic attribute types INPUT, OUTPUT, and TEXTGE. Dynamic area attribute types are described in more detail in "Dynamic areas" on page 62.

CUA Can be used if you are creating panels according to the CUA guidelines. CUA panel types are described in more detail in "CUA attribute types" on page 50.

The third column of each table indicates whether a particular value of the TYPE keyword can be coded as a variable. For example:

- You must specify TYPE(TEXT) directly.
- For another attribute, you can specify TYPE(&VAR) and let the dialog program set the value of VAR to either INPUT or OUTPUT. The dialog program cannot set the value of VAR to TEXT.

Value	Description	Variable?
TEXT	Text (protected) field	No
INPUT	Input (unprotected) field	Yes
OUTPUT	Output (protected) field	Yes
TEXTGE	Text (protected) field, optionally preceded with graphic escape characters: E Top left-hand corner N Top right-hand corner D Bottom left-hand corner M Bottom right-hand corner s Horizontal line e Vertical line F Connector left O Connector right G Connector up P Connector down L Intersection	No

Figure 28. Basic attribute types

Value	Description	Variable?
DATAIN	Input (unprotected) field in a dynamic area	Yes
DATAOUT	Output (protected) field in a dynamic area	Yes
DATAOUTGE	Output (protected) field in a dynamic area, optionally preceded with graphic escape characters: E Top left-hand corner N Top right-hand corner D Bottom left-hand corner M Bottom right-hand corner s Horizontal line e Vertical line F Connector left O Connector right G Connector up P Connector down L Intersection	No

Figure 29. Dynamic area attribute types

Value	Description	Variable?
AB	Action-bar choice	No
ABSL	Action-bar separator line	No
CEF	Choice entry field	Yes
CH	Column heading	No
CT	Caution text	No
DT	Descriptive text	No
EE	Error emphasis	Yes

Value	Description	Variable?
EEF	Emphasized entry field	Yes
ET	Emphasized text	No
FP	Field prompt	No
LEF	List entry field	Yes
LI	List items	Yes
LID	List-item description	Yes
MSGA	Action message text	Yes
MSGI	Information message text	Yes
MSGS	Short message text	No
MSGW	Warning message text	Yes
NEF	Normal entry field	Yes
NT	Normal text	No
PDSL	Pull-down separator line	No
PIN	Panel instruction	No
PT	Panel title	No
RP	Reference phrase	No
SAC	Select available choices	Yes
SE	Selected emphasis	Yes
SF	Selection field	No
SI	Scroll information	No
SUC	Select unavailable choices	Yes
VOI	Variable output information	Yes
WASL	Work-area separator line	No
WT	Warning text	No

Figure 30. CUA attribute types

Other keywords

In addition to the TYPE keyword, you can specify the following keywords. You can specify the value directly or as a variable.

COLOR(WHITE|RED|BLUE|GREEN|PINK|YELLOW|TURQ)

For terminals that support the IBM 3270 extended data stream, the COLOR keyword defines the color of a field. (TURQ means turquoise.) If a color is not specified and the panel is displayed on a color terminal, a default color is generated based on the protection (TYPE) and intensity attributes of the field. Figure 31 shows the default colors.

Field type	Intensity	Default color
Text or Output	HIGH	WHITE
Text or Output	LOW	BLUE
Input	HIGH	RED
Input	LOW	GREEN

Figure 31. Default colors

INTENS(HIGH|LOW|NON)

The intensity of the field:

HIGH	High intensity field (the default)
LOW	Low (normal) intensity field
NON	Nondisplay field

You can specify these operands for input or output fields. The NON operand lets the dialog control whether text is displayed. For example, in Figure 32 on page 46 the dialog controls whether the instructional information is displayed. With variable &MYVALUE set to NON, the three instruction lines defined by the ¢ attribute character are not displayed. With &MYVALUE set to LOW, for inexperienced users, these lines are displayed at low intensity.

```
)ATTR DEFAULT (%+_)
  ¢ TYPE(TEXT) INTENS(&MYVALUE)
  @ TYPE(INPUT) INTENS(LOW)
)BODY
+NLDATAIN          %--- NEW LEAD DATA ENTRY PANEL ---
%COMMAND ===>_ZCMD
%
+  ENTER THE NAME OF A NEW CUSTOMER LEAD FOR SALES
+  REPRESENTATIVE:@SALEREP
%
  CUSTOMER NAME:
    FIRST NAME    %===>@FNAME
    MIDDLE INITIAL%===>@Z
    LAST NAME     %===>@LNAME
%
¢    ENTER THE CUSTOMER'S FIRST NAME, MIDDLE INITIAL, AND LAST NAME.
¢    PRESS THE ENTER KEY TO STORE THE CUSTOMER DATA AS ENTERED ABOVE.
¢    PRESS THE END KEY (PF3) TO END THIS SESSION.
%
)INIT
  .ZVARS = MINIT
  IF (&USER = 'EXPERIENCED')
    &MYVALUE = NON
  ELSE
    &MYVALUE = LOW
)PROC
      VER (&FNAME,NB,ALPHA,MSG = MSG881)
      VER (&MINIT,ALPHA,MSG = MSG882)
      VER (&LNAME,NB,ALPHA,MSG = MSG883)
      VPUT (FNAME MINIT LNAME) PROFILE
)END
```

Figure 32. Panel definition illustrating use of INTENS(NON)

HILITE(USCORE|BLINK|REVERSE)

For SDF2/DM-supported terminals with the extended highlighting feature, the extended highlighting attribute for a field:

USCORE	Underscore
BLINK	Blinking
REVERSE	Reverse video

No default is assumed if highlighting is not specified.

This keyword is ignored if the panel is displayed on a terminal without the extended highlighting feature.

CUATYPE(*value*)
Specifies that a non-CUA attribute type (type TEXT, TEXTGE, INPUT, OUTPUT, DATAIN, DATAOUT, or DATAOUTGE) should have the same COLOR, INTENSITY, and HILITE values as the specified CUA attribute type. Any COLOR, INTENSITY, and HILITE values that you specify are ignored.

For example, if you specify:

```
¬ TYPE(TEXT) CUATYPE(WT)
```

Any text that is marked with the ¬ symbol will have the same color, intensity, and highlighting as CUA warning text (WT). By default, the text will be white, high intensity, and not highlighted. If a user changes CUA warning text to red (with the CUAATTR command), the text marked with the ¬ symbol will also change to red.

CAPS(ON|OFF|IN|OUT)
Specifies the uppercase or lowercase attribute of a field. CAPS is not valid for text fields. The CAPS keyword can have the following values:

ON	Data is translated to uppercase before being displayed and all input fields are translated to uppercase before being stored.
OFF	Data is displayed as it appears in the function pool or shared pool, and the contents of all input fields are stored as they appear on the screen.
IN	Data is displayed as it appears in the function pool or shared pool, but all input fields on the screen are translated to uppercase before being stored.
OUT	Data is translated to uppercase before being displayed. All input fields are stored as they appear on the screen.

If you omit the CAPS parameter, the default is:

- CAPS(OFF) for DATAIN, DATAOUT, and DATAOUTGE fields in dynamic areas
- CAPS(ON) for all other input or output fields

JUST(LEFT|RIGHT|ASIS)
Specifies how the contents of the field are justified when displayed. JUST is valid only for input and output fields.

LEFT	Left justification
RIGHT	Right justification
ASIS	No justification

Justification occurs if the initial value of a field is shorter than the length of the field as described in the panel body. Normally, right justification should be used only with output fields, because a right-justified input field would be difficult to overtype.

For LEFT or RIGHT, the justification determines only how the field appears on the screen. Leading blanks are automatically deleted when the field is processed. For ASIS, leading blanks are not deleted when the field is processed or when it is initialized. Trailing blanks are automatically deleted when a field is processed, regardless of its justification.

If you omit the JUST parameter, the default is:

- JUST(ASIS) for DATAIN, DATAOUT, and DATAOUTGE fields in dynamic areas
- JUST(LEFT) for all other input or output fields

PAD(NULLS|*char*)
Specifies the pad character for initializing the field. This is not valid for text fields.

NULLS	Nulls are used for padding.
char	Any character, including blank (' '), can be specified as the padding character. If the character is any of the following, it must be enclosed in single quotes:

```
blank < ( + ) ; ¬ , > : =
```

If the desired pad character is a single quote, use four single quotes: `PAD('''')`

If the field is initialized to blanks, or if the corresponding dialog variable is blank, the entire field contains the pad character when the panel is first displayed. If the field is initialized with a value, any remaining field positions contain the pad character.

Padding and justification work together, as follows. At initialization, unless you have specified ASIS, the field is first justified, then padded. For left-justified and ASIS fields, the padding extends to the right. For right-justified fields, the padding extends to the left.

When SDF2/DM processes an input field, it automatically deletes leading or trailing pad characters as follows:

- For a left-justified field, SDF2/DM deletes leading and trailing pad characters.
- For a right-justified field, SDF2/DM deletes leading pad characters and stores trailing pad characters.
- For an ASIS field, SDF2/DM deletes trailing pad characters and stores leading pad characters.
- Regardless of the type of justification, SDF2/DM deletes leading and trailing pad characters for command fields.

SDF2/DM never deletes imbedded pad characters. It deletes only leading or trailing pad characters.

PADC(<u>NULLS</u>|*char*)
Specifies conditional padding with the specified pad character. The pad character is used as a field filler only if the value of the input or output field is initially blank. The pad character is not displayed in the remaining unfilled character positions if the field has an initial value. Instead, the unfilled positions contain nulls. Otherwise, SDF2/DM treats the PADC keyword like the PAD keyword, including justification and deletion of pad characters before storing variables.

If the PAD and PADC parameters are omitted, the default is PAD(NULLS) for input and output fields.

SKIP(ON|<u>OFF</u>)
The SKIP keyword defines the autoskip attribute of the field. It is valid only for text or output (protected) fields. SKIP(OFF) is the default.

SKIP(ON)	Specifies that the cursor is to automatically skip the field. When a character is entered into the last character location of the preceding unprotected field, SDF2/DM positions the cursor at the first character location of the next unprotected field.
SKIP(OFF)	Specifies that the cursor is not to automatically skip the field when a character is entered into the last character position of a field.

NUMERIC(ON|<u>OFF</u>)

For terminals with the Numeric Lock feature, the NUMERIC attribute keyword allows users to be alerted to certain keying errors.

ON	Activates the Numeric Lock feature. The terminal keyboard locks if the operator presses any key other than 0 through 9, minus(-), period (.), or duplicate (DUP). On non-English keyboards, the period may be replaced by a comma as a valid numeric character. ON is valid only for unprotected fields.
OFF	Specifies that the Numeric Lock feature is not to be activated. The terminal operator can key in any characters. This is the default.

On a data-entry keyboard with the Numeric Lock feature, when the user moves the cursor into a field defined by the NUMERIC (ON) attribute keyword, the display shifts to numeric mode. If the user presses any key other than those allowed by the Numeric Lock feature, the `DO NOT ENTER` message is displayed in the operator information area and the terminal is disabled. The user can continue by pressing the reset key.

NUMERIC(ON) and SKIP(ON) attributes cannot be specified for the same field.

FORMAT(<u>EBCDIC</u>|DBCS|MIX)

For DBCS terminals, the FORMAT keyword specifies the character format for a field.

EBCDIC	EBCDIC characters only
DBCS	DBCS characters only
MIX	EBCDIC and DBCS characters

In a FORMAT(MIX) field, any DBCS character string must be enclosed by a shift-out code (hexadecimal 0E) and a shift-in code (hexadecimal 0F).

The default value for a TYPE(INPUT) and TYPE(DATAIN) field is FORMAT(EBCDIC).

The default value for a TYPE(TEXT), TYPE(TEXTGE), TYPE(OUTPUT), TYPE(DATAOUT), and TYPE(DATAOUTGE) field is FORMAT(MIX).

The pad character for a DBCS field is converted to the corresponding 16-bit character, and is then used for padding. Other format fields are padded normally.

The CAPS attribute is meaningful only for EBCDIC and MIX fields. In addition, within a MIX field, the CAPS attribute applies only to the EBCDIC subfields.

CUA attribute types

Figure 33 shows the default values for each CUA attribute type. You can override all the defaults, except the COLOR, INTENS, and HILITE values.

CUA attribute types can be described in terms of basic attribute types, as follows:

Basic type	**CUA attribute types**
Input, Unprotected	CEF, EE, EEF, LEF, NEF
Output, Protected	LI, LID, MSGA, MSGI, MSGS, MSGW, VOI
Text, Protected	ABSL, CH, CT, DT, ET, FP, NT, PDSL, PIN, PT, SAC, SE, SI, SUC, WASL, WT
Text, Unprotected	AB, RP, SF

Figure 33. CUA TYPE keyword values

Value	COLOR	INTENS	HILITE	CAPS	JUST	PAD	PADC	SKIP	NUMERIC	FORMAT
AB	WHITE	HIGH	NONE	n/a	n/a	n/a	n/a	n/a	n/a	MIX
ABSL	BLUE	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
CEF	TURQ	LOW	USCORE	OFF	LEFT		6D	n/a	OFF	EBCDIC
CH	BLUE	HIGH	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
CT	YELLOW	HIGH	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
DT	GREEN	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
EE	YELLOW	HIGH	REVERSE	OFF	LEFT		6D	n/a	OFF	EBCDIC
EEF	WHITE	HIGH	USCORE	OFF	LEFT			n/a	OFF	EBCDIC
ET	TURQ	HIGH	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
FP	GREEN	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
LEF	TURQ	LOW	USCORE	OFF	ASIS		6D	n/a	OFF	EBCDIC
LI	WHITE	LOW	NONE	OFF	ASIS	40		OFF	n/a	MIX
LID	GREEN	LOW	NONE	OFF	ASIS	40		OFF	n/a	MIX
MSGA	RED	HIGH	NONE	n/a	n/a	n/a	n/a	n/a	n/a	MIX
MSGI	WHITE	HIGH	NONE	n/a	n/a	n/a	n/a	n/a	n/a	MIX
MSGS	YELLOW	HIGH	NONE	n/a	RIGHT	n/a	n/a	ON	n/a	MIX
MSGW	YELLOW	HIGH	NONE	n/a	n/a	n/a	n/a	n/a	n/a	MIX
NEF	TURQ	LOW	USCORE	OFF	LEFT		6D	n/a	n/a	EBCDIC
NT	GREEN	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
PDSL	BLUE	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
PIN	GREEN	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
PT	BLUE	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
RP	TURQ	HIGH	NONE	n/a	n/a	n/a	n/a	n/a	n/a	MIX
SAC	WHITE	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
SE	YELLOW	HIGH	NONE	OFF	ASIS	n/a	n/a	n/a	OFF	EBCDIC
SF	WHITE	LOW	NONE	n/a	n/a	n/a	n/a	n/a	n/a	MIX
SI	WHITE	HIGH	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
SUC	BLUE	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
VOI	TURQ	LOW	NONE	OFF	LEFT	40		OFF	n/a	MIX
WASL	BLUE	LOW	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX
WT	WHITE	HIGH	NONE	n/a	n/a	n/a	n/a	OFF	n/a	MIX

Notes:

1. n/a (not applicable) means that the keyword cannot be used with this TYPE value.
2. A value can be set for PAD or PADC but not both. For LI, LID, and VOI, PAD is blank (X'40'). For CEF, EE, LEF, and NEF, PADC is underscore (X'6D') if the terminal does not support HILITE.
3. You cannot change the value of the COLOR, INTENS, or HILITE keyword for a CUA attribute type in the)ATTR section of a panel definition. Users can change the values of these keywords interactively, using the CUAATTR command.

Figure 34 on page 51 shows TYPE values that you cannot specify in the)ATTR section but can specify with the .ATTR system variable. For example, with the .ATTR system variable you can temporarily assign TYPE(PDUC) to a pull-down choice that is not currently available.

Panel element attribute	Value	COLOR	INTENS	HILITE
Action-bar unavailable choice	ABUC	BLUE	LOW	NONE
Pull-down available choice	PD	WHITE	LOW	NONE
Pull-down unavailable choice	PDUC	BLUE	LOW	NONE

Figure 34. Additional CUA attributes for .ATTR

Figure 35 lists CUA panel-element attributes that are used *internally* by SDF2/DM in response to user interactions. These attributes do not have a TYPE keyword, so you cannot specify them in the)ATTR section or the .ATTR system variable. They are considered protected text fields.

Users can change the color, intensity, and highlighting interactively by using the CUAATTR command.

Panel element attribute	COLOR	INTENS	HILITE
Action-bar selected choice	YELLOW	LOW	NONE
Function keys	BLUE	LOW	NONE
Panel ID	BLUE	LOW	NONE
Background border	BLUE	HIGH	NONE
Pop-up border	YELLOW	HIGH	NONE
Help pop-up border	YELLOW	HIGH	NONE

Figure 35. Internal attributes without TYPE values

The)ABC section

The)ABC (action-bar choice) section is optional. It defines an action-bar choice for a panel and its associated pull-down choices. An)ABC section must exist for each action-bar choice displayed in the action-bar area on a panel.

The)ABC header statement

The)ABC header statement has the following syntax:

Syntax

```
)ABC DESC(choice-description-text) [NAME(field-name)]
                                   [MNEMPOS(w)]
```

DESC(*choice-description-text***)**

Text displayed in the panel's action-bar area for the action-bar choice. The maximum length of the text is 64 characters. Each action-bar choice must be unique. The *choice-description-text* can contain variable names, which are substituted for text when the panel is displayed.

The *choice-description-text* must match the text specified in the)BODY section of the panel.

If the text contains any special characters or blanks, enclose it in quotes in the)ABC DESC parameter, but do not enclose it in quotes in the)BODY section.

NAME(*field-name***)**
The name of the action-bar choice. This name can be referenced in the Help section, to provide help for the action-bar choice. The name can contain from 1 to 8 characters.

If you omit the NAME parameter, the name used is **ZABC***xx*, where *xx* is the number of the action-bar choice. Counting starts with 01.

MNEMPOS(*w***)**
The position within the choice-description text of the mnemonic character (an underlined character that the user can enter to select a choice). Counting starts with 1.

If you are using mnemonic characters for action-bar choices, you should put an entry field for mnemonic characters at the start of the action-bar line in the)BODY section.

Defining pull-down choices

When a user selects an action-bar choice, a pull-down can appear. Within the)ABC section, define each pull-down choice as follows:

1. Code a PDC statement.
2. Immediately after the PDC statement, code an ACTION statement that specifies what happens when the pull-down choice is selected.

The PDC statement has the following syntax:

Syntax

PDC DESC(*choice-description-text***)** [**NAME(***field-name***)**]

DESC(*choice-description-text***)**
Actual text that is displayed for the pull-down choice.

The text can contain a variable name, which is substituted for text when the panel is displayed. Special characters or blanks must be enclosed within quotes. The maximum length of the text is 64 characters. Do not include choice numbers in your text. SDF2/DM numbers each pull-down choice in the)ABC section sequentially, starting with 1, and displays the number before the *choice-description-text*.

Use the ZTAB system variable to align text like the accelerator key text on the right side of the pull-down menu. For example:

```
PDC DSCRPT('Exit &ZTAB.F3')
```

NAME(*field-name***)**
The name of the pull-down choice. This name can be referenced in the Help section to provide help for the pull-down choice. The name can contain from 1 to 8 characters.

If no name is specified, the name **ZPDC***xxyy* is used, where *xx* is the number of the action-bar choice and *yy* is the number of the pull-down choice within this action-bar choice. Counting starts with 01.

The ACTION statement has the following syntax:

Syntax

```
ACTION  [RUN(command-name)]
        [PARM(command-parms)]
        [PANEL(panel-name)]
```

RUN(*command-name***)**
Specifies the name of a command to be executed. The *command-name* must be 1–8 characters. To specify a variable, precede the name with an ampersand (&).

You must specify **RUN(***command-name***)** or **PANEL(***panel-name***)**, or both.

PARM(*command-parms***)**
Parameters to be used when processing the command. Enclose the *command-parms* value in quotes if it contains special characters or blanks. This keyword applies only if you specify **RUN(***command-name***)**.

PANEL(*panel-name***)**
The panel to be displayed when the pull-down choice is selected. This can be used for cascading pull-downs or additional parameter gathering. To specify a variable, precede the name with an ampersand (&).

You can define only one ACTION statement for each PDC statement in the)ABC panel section. Specify the keywords in any order. If a keyword is duplicated within an ACTION statement, SDF2/DM uses the last occurrence of the keyword.

Adding pull-down separator lines

If some of the choices on a pull-down are to be grouped together, you can delineate the groups by adding one or more separator lines to the pull-down. To do this, add a PDCSEP statement after one PDC statement (and its associated ACTION statement) but before the next PDC statement.

The PDCSEP statement has the following syntax:

Syntax

```
PDCSEP  [DESC(separator-text)]
        [RULE]
        [LEFT | CENTER | RIGHT]
```

DESC(*separator-text***)**
Optional text that is displayed on the separator line. Enclose special characters or blanks within quotes.

RULE
Leading and trailing blanks in the separator line are replaced by a straight horizontal line on a screen with graphic capabilities; otherwise, the space is filled with dots. If no separator text is specified, the entire line is filled with a rule.

LEFT
The separator text will be left-adjusted. The keywords LEFT, CENTER, and RIGHT are mutually exclusive.

CENTER
The separator text will be centered. The keywords LEFT, CENTER, and RIGHT are mutually exclusive.

RIGHT
The separator text will be right-adjusted. The keywords LEFT, CENTER, and RIGHT are mutually exclusive.

Defining an action-bar choice with no pull-down

In some cases, selecting an action-bar choice causes a pull-down to appear; in other cases, selecting an action-bar choice causes an immediate action.

If you do not want a pull-down for an action-bar choice, do not code any PDC statements in the)ABC section. The)ABC section then consists entirely of an)ABC header statement. When a user selects this action-bar choice, the)ABCPROC section gets control. In the)ABCPROC section, you can set the system variable ZCMD to the appropriate command.

The)ABCINIT section

The)ABCINIT section is optional. If you include it, it must follow an)ABC section.

The)ABCINIT header statement has no parameters.

The)ABCINIT section contains initialization instructions that are performed when a user selects the action-bar choice, before the pull-down menu is displayed. The)ABCINIT section can contain the same instructions as the)INIT section. For more information, see “The)INIT section” on page 84.

When the pull-down menu for this action-bar choice is displayed, it will contain an entry field. The variable name used for this entry field is ZPDC.

The)ABCPROC section

The)ABCPROC section is optional.

After each)ABCINIT section, or immediately after the)ABC section if no)ABCINIT section is coded, you can code an)ABCPROC section. The)ABCPROC header statement has no parameters.

The)ABCPROC section contains processing instructions that are performed when a user finishes interacting with the pull-down menu. The)ABCPROC section can contain the same instructions as the)PROC section. For more information, see “The)PROC section” on page 86.

If the)ABC section consists solely of an)ABC header statement, the)ABCPROC section gets control when the action-bar choice is selected.

Specifying the action bar in the)ATTR and)BODY sections

In the)ATTR section, specify an attribute character with type AB (action-bar choice). You can optionally specify an attribute character with type NT (normal text).

In the)BODY section, specify the appearance of the action bar. The action bar must immediately follow the)BODY statement. The action bar can take up more than one line, if necessary.

In the)BODY section, do the following:

1. If you want an entry field that can be used for mnemonic characters, put it at the start of the first action-bar line. It should have TYPE(INPUT).
2. After the entry field (if any), type the AB attribute character followed by a blank.
3. Type the text for the first action-bar choice. This text must exactly match the text on the corresponding DESC keyword of the)ABC statement, except that the text must not be enclosed in quotes.
4. For each remaining action-bar choice, type the AB character, a blank, and the description text.
5. After the last action-bar choice on each line, type a text attribute character (for example, the NT attribute character if you defined one).
6. Leave a blank line after the last action-bar line. In this line, SDF2/DM automatically puts an action-bar separator line.

Internally, action-bar choices are numbered sequentially from left to right and from top to bottom starting with 1.

A one-line action bar is coded like this:

```
)ATTR
 @ TYPE(AB)
)BODY
 @ choice1@ choice2@ choice3+
```

A multiline action bar is coded like this:

```
)ATTR
 @ TYPE(AB)
)BODY
 @ choice1@ choice2@ choice3+
 @ choice4@ choice5@ choice6+
```

Example

```
)PANEL
)ATTR
 @ TYPE(AB)
 # TYPE(NT)
  ⋮
)ABC DESC(FILE)
  PDC DESC(file-choice1)
  ACTION RUN(command1) PARM(parms1)
  PDC DESC(file-choice2)
  ACTION RUN(command2) PARM(parms2)
  PDC DESC(file-choice3)
  ACTION RUN(command3) PARM(parms3)
)ABC DESC(HELP)
  PDC DESC('Help for help')
  ACTION RUN(HELP) PARM(DGIRHELP)
  PDC DESC('Extended help')
  ACTION RUN(EXHELP)
  PDC DESC('Keys help')
  ACTION RUN(KEYSHELP)
  PDC DESC('Help index')
  ACTION RUN(INDEX)
  PDC DESC('Tutorial')
  ACTION RUN(HELP) PARM(&ZHTOP)
)BODY
 @ FILE@ HELP#
  ⋮
)END
```

Figure 36. Action-bar example

The)BODY section

The)BODY (panel body) section is required. It specifies the format of the panel as the user sees it. Each record in the body section corresponds to a line on the display.

If there are no preceding sections and you do not want to specify any keywords on the)BODY header statement, you can omit the)BODY header statement. The first line of the panel definition is then considered the first line of the)BODY section.

The)BODY header statement and all associated keywords must be specified on the same line. The panel body ends with any of the following statements:

)AREA
)INIT
)REINIT
)PROC
)HELP
)KEYLIST
)END

The)BODY header statement has the following syntax:

Syntax

```
)BODY [DEFAULT(def1def2def3)]
      [WIDTH(width)]
      [EXPAND(xy)]
      [WINDOW(width,depth)]
      [CMD(field-name)]
      [SMSG(field-name)]
      [LMSG(field-name)]
      [CONTCHAR(c)]
```

DEFAULT(*def1def2def3***)**
Specifies the characters that define a high-intensity text field (*def1*), a low-intensity text field (*def2*), and a high-intensity input field (*def3*). The value inside the parentheses must consist of exactly three characters, not enclosed in single quotes and not separated by commas or blanks.

The DEFAULT keyword can be specified on either the)ATTR header statement or the)BODY header statement. If it is specified on both, the)BODY specification takes precedence.

For more information, see “The)ATTR section” on page 41.

WIDTH(*width***)**
The number of columns to use in formatting the panel. It can be a constant or a dialog variable, including the system variable ZSCREENW. The specified width must not be less than 80 or greater than the width of the terminal on which the panel is to be displayed. If the WIDTH keyword is not specified, the default *width* value 80 is used.

If the panel definition width is greater than 80 characters, the WIDTH keyword must be used. If the WIDTH keyword is used, the WIDTH variable must be set in the function pool or shared pool before the panel is displayed.

EXPAND(*xy***)**
The repetition delimiter characters. The delimiters can be used on any line within the panel body to enclose a single character that is repeated to expand the line to the required width. The starting and ending delimiter can be the same character. If no delimiters are specified, or if any line does not contain the delimiters, the line is expanded to the required width by adding blanks on the right. The delimiter characters *cannot* be specified with a dialog variable.

WINDOW(*width***,***depth***)**
Defines the width and depth of the pop-up window that the dialog manager uses to display the panel. The values do not include the panel borders; the dialog manager adds them outside of the dimension of the width and depth values. If the text on the panel you are defining exceeds the width of the window, the panel fields do not wrap. All fields end at the window width.

The width that you specify must be a numeric value greater than or equal to the minimum width of 8 characters. The depth that you specify must be a numeric value greater than 0.

The width and depth cannot be specified by a dialog variable.

For panels that are not to be displayed in a pop-up window (the ADDPOP service has not been requested), SDF2/DM validates the width and depth values against the screen size and issues an error message if either of the following is true:

- The window width is greater than the current device width.
- The window depth is greater than the current device depth.

For panels that are being displayed in a pop-up window (after the ADDPOP service has been requested), SDF2/DM validates the width and depth values against the screen size minus the frame and issues an error message if either of the following is true:

- The window depth is greater than the screen depth minus 2.
- The window width is greater than the screen width minus 4.

If the panel is not being displayed in a pop-up window, SDF2/DM validates the keyword, but ignores it.

CMD(*field-name***)**
Identifies the panel field (variable name) to be treated as the command field. The field must be TYPE(INPUT). The default is ZCMD.

SMSG(*field-name***)**
Identifies the panel field (variable name) where the short message, if any, is to be placed. The TYPE must be OUTPUT or MSGS. If the message is longer than the length of this field, the subsequent field is overlayed with the message text.

LMSG(*field-name***)**
Identifies the panel field (variable name) where any long message is to be placed. The TYPE must be OUTPUT, MSGA, MSGI, or MSGW. If the message is longer than the length of this field, the message is placed in a pop-up window.

Note: For CMD, SMSG, and LMSG the field-name must be within the)BODY section, not within a scrollable area.

CONTCHAR(*single-character***)**
Specifies a single character, which is used as a continuation character in the)BODY section. The character must be the last character of the record. Any character except `X'00'` is allowed. The next record is then concatenated with current record, starting at the position of the continuation character.

Statements within the)BODY section

After the)BODY header statement, you include a picture of the panel that you are defining. Use attribute characters to indicate how different sections of the panel are to be displayed. (Attribute characters are defined in the)ATTR section and in the DEFAULT keyword of the)ATTR or)BODY header statement.)

Identify different types of fields as follows:

- Before the start of a text field, specify a text attribute character. The default attribute characters are % for high-intensity text and + for low-intensity text.

  If you want to include a variable within a text field, specify the variable name preceded by an ampersand. When the panel is displayed, the variable name will be replaced by its current value with trailing blanks stripped.

- Before the start of an input field, specify an input attribute character. The default attribute character is _ for a high-intensity input field.

  Immediately after the attribute character, specify a variable name *without* an ampersand. No text can be included within an input field. (Because an input field or output field consists entirely of a variable, the terms *field* and *dialog variable* are used interchangeably.)

  If the variable name is longer than the input field, type the placeholder variable name Z. In the)INIT section of the panel definition, use the .ZVARS control variable to specify the actual variable names for all Z variables. For more information about .ZVARS, see Chapter 4, “Control variables” on page 100.

  SDF2/DM initializes input fields before the panel is displayed. The user can then type over any initial value.

- Before the start of an output field, specify an output attribute character. There is no default attribute character for output fields.

  As with an input field, the attribute character for an output field is followed by a variable name without an ampersand. The placeholder variable name Z can also be used.

  SDF2/DM initializes output fields before the panel is displayed. The user cannot change the value of an output field.

- For a selection field, follow the instructions in “Selection fields” on page 61.
- For a scrollable area, follow the instructions in “The)AREA section” on page 69.
- For a dynamic area, follow the instructions in “Dynamic areas” on page 62.

Controlling panel width

Before a panel is displayed, it is formatted according to the WIDTH and EXPAND keyword values as if the “expanded format” of the body were originally coded in the panel definition. For example:

```
)BODY  WIDTH(&EDWIDTH) EXPAND(//)
+-- &TITLE --------------------------------/-/----------
%COMMAND ===>_ZCMD     / /            +SCROLL%===>_SCRL +
+
%EMPLOYEE NUMBER:@EMPLN        / /                    @
```

In the title line, hyphens are repeated to expand the line to the width specified by &EDWIDTH. The command field and the employee number field would both be expanded with repeated blanks.

If more than one repetition character appears in a line of the panel body, each of the characters is repeated an equal number of times. For example:

```
)BODY  EXPAND(#@)
TUTORIAL #-@ TITLE OF PAGE #-@ TUTORIAL
```

would become:

```
TUTORIAL ------------ TITLE OF PAGE ------------ TUTORIAL
```

SDF2/DM treats as an error any request to display a panel that is wider than the physical screen. SDF2/DM displays a “box” panel indicating the error.

Figure 37 on page 60 shows all the valid combinations of WIDTH and WINDOW values. For each combination, it shows the width of the expanded panel if the EXPAND keyword is used, and the dimensions of the pop-up window if the WINDOW keyword is specified for a pop-up window.

Figure 37. Combinations of WIDTH and WINDOW values

WINDOW	**WIDTH**	**Expansion**	**Pop-up window**
none	none	80	(76, 22)
w≤80	none	WINDOW *w*	WINDOW (*w*, *d*)
none	*value*≥80	WIDTH *value*	(76, 22)
w≤WIDTH	*value*≥80	WINDOW *w*	WINDOW (*w*, *d*)

Note: SDF2/DM will issue an error message if you attempt to display a panel in a pop-up window where the WINDOW width value is greater than the width of the underlying panel.

Variable substitution

When the panel is displayed, each variable name and its preceding ampersand are replaced by the current value of the variable with trailing blanks stripped. Regardless of the length of the variable, the length of the field that it appears in does not change.

If the variable value is longer than the variable name and preceding ampersand, characters are removed from the field starting at the right. If the variable value is shorter, the rightmost character is repeated as many times as necessary, provided that it is not a letter or number. (The end of the field is the next attribute character.)

For example:

```
%DATA SET NAME: &DSNAME ----------------------%
```

could be displayed like this:

```
 DATA SET NAME: X.Y.Z ------------------------%
```

or this:

```
 DATA SET NAME: FIRST.SECOND.THIRD -----------%
```

Fields defined in a scrollable area end at the end of the line where their definition starts. They will not wrap to the next line.

If you use the EXPAND keyword on the)BODY header statement, expanded lines are expanded before variables are substituted. If a variable on an expanded line is longer than the variable name, meaningful text at the right of the line may be lost.

For example:

```
)BODY  EXPAND(//)
TUTORIAL /-/ &VAR1 /-/ TUTORIAL
```

would become:

```
TUTORIAL ---------------- &VAR1 ---------------- TUTORIAL
```

If VAR1 has the value ABCDEFG when the screen is displayed, the line becomes:

```
TUTORIAL ---------------- ABCDEFG ---------------- TUTORI
```

To avoid this problem, provide a few blanks before the attribute character that ends the text line. For example:

```
TUTORIAL /-/ &VAR1 /-/ TUTORIAL      +
```

Selection fields

A selection field is a text field that the user can move to with the tab keys and select with the cursor. Selection fields are appropriate if the user must select one option from a predefined list of options.

To define a list of selection fields, do the following:

- In the)ATTR section, assign an attribute character with TYPE(SF).
- In the)BODY and)AREA sections, mark the beginning of each selection field with the SF attribute character and mark the end of each selection field with a text attribute character.
- The selection field may contain a variable name. In this case the text is substituted when the panel is displayed.
- The field name for the selection field is ZSF*xxyyy*, where:

  xx Is 00 if the selection field appears in the)BODY section, or a number from 01 to 99 if it appears in an)AREA section. The area sections are numbered, based on where they appear within the panel body. Numbering starts at the top left-hand corner of the panel and continues through each line of the panel from left to right.

  yyy Is the number of the selection field within its)BODY or)AREA section, from 001 to 999. Numbering starts at the top left-hand corner of the)BODY or)AREA section and continues through each line of the section from left to right.

You can use the .ZSF control variable to assign meaningful variable names to selection fields.

For example:

```
)PANEL
)ATTR
  _ TYPE(CEF)
  $ TYPE(SF)
)BODY
_Z$1. ON +
  $2. OFF+
)INIT
 .ZVARS=SELFLD
 &SELFLD=''
 &ZWINTTL=''
)PROC
 IF(&SELFLD='')
   &OPT= TRANS(.CURSOR ZSF00001,'ON' ZSF00002,'OFF', *,'')
 ELSE
   &OPT= TRANS(&SELFLD 1,'ON' 2,'OFF',MSG=DGIM0006)
)KEYLIST
 key(f1)  cmd(help)     fka(no )      text('')
 key(f3)  cmd(exit)     fka(no )      text('')
 key(f12) cmd(cancel)   fka(no )      text('')
)END
```

Dynamic areas

To define a dynamic area:

1. In the)ATTR section, specify an attribute character that defines the borders of the dynamic area. Specify also whether the dynamic area is to expand to fill all the available space on the screen, whether it is to be scrollable, and which codes are to be set when a field in the dynamic area is modified.
2. In the)BODY section, use the attribute character to mark the place in the panel where the dynamic area will be displayed.
3. In your dialog program, put the logic that determines the contents, format, and processing of the dynamic area.

The format of a dynamic area is specified by a string of control and data characters, stored in a dialog variable. This variable was produced either in the current dialog or, earlier, in another dialog or program. The string usually contains a mixture of nondisplayable attribute characters and data to be displayed. The name of the dialog variable is chosen by the panel designer. This name is placed in the dynamic area of the panel definition.

A dialog uses the DISPLAY service to display a panel containing a dynamic area. After the display and after entry of any input by the user, data from within the dynamic area is stored in the variable associated with the area and is available for processing by the application prototype.

You can specify more than one dynamic area on a panel. The number of dynamic areas in a panel definition is limited only by physical space limitations of the particular terminal being used for the display.

The)ATTR section

In the)ATTR section, use TYPE(DATAIN) instead of TYPE(INPUT) for unprotected fields within the dynamic area. Similarly, use TYPE(DATAOUT) or TYPE(DATAOUTGE) instead of TYPE(OUTPUT) or TYPE(TEXTGE) for protected fields within the dynamic area. These fields have the following defaults:

DATAIN	CAPS(OFF) JUST(ASIS) PAD(' ')
DATAOUT	CAPS(OFF) JUST(ASIS) PADC(' ')
DATAOUTGE	CAPS(OFF) JUST(ASIS) PADC(' ')

For example, you could have the following lines in the)ATTR section for input and output fields within the dynamic area:

```
21 TYPE(DATAIN)  CUATYPE(SE)  CAPS(ON) /* selected item          */
22 TYPE(DATAIN)  CUATYPE(EE)  CAPS(ON) /* error emphasis         */
23 TYPE(DATAIN)  CUATYPE(VOI)          /* selectable output      */
24 TYPE(DATAOUT) CUATYPE(VOI)          /* variable output info   */
25 TYPE(DATAOUT) CUATYPE(EE)           /* error emphasis         */
26 TYPE(DATAOUT) CUATYPE(ET)           /* emphasized text        */
```

In the)ATTR section, specify an attribute character. You can use the same attribute character for several dynamic areas if the dynamic areas all have the same EXTEND, SCROLL, USERMOD, and DATAMOD values.

Use the following syntax:

Syntax

)ATTR *char* **AREA(DYNAMIC)** [**EXTEND(ON|<u>OFF</u>)**]
[**SCROLL(ON|<u>OFF</u>)**]
[**USERMOD(***usermod-code***)**]
[**DATAMOD(***datamod-code***)**]

char **AREA(DYNAMIC)**

Specifies the special character that marks out the dynamic area within the panel body section. You can specify either the character itself or the 2-digit hexadecimal number that corresponds to the character.

In the panel body section, the name immediately following this character identifies the dialog variable that contains the dynamically formatted string to be displayed in the area. Subsequent lines of the dynamic area are defined in the panel body by placing this character in the starting and ending columns of the dynamic area. Except on the first line of the dynamic area, where the area name immediately follows the left-hand delimiter character, at least one blank must follow the delimiter characters on the left-hand side of the dynamic area.

This is a special character, not an actual attribute character. You cannot define other fields within or overlapping a dynamic area.

EXTEND(ON|<u>OFF</u>)

EXTEND(ON) specifies that the depth of the dynamic area is automatically increased so that panel fills the physical screen on which it is displayed. In the)BODY section, you need provide only the first line of the dynamic area, unless other text or fields are to be displayed to the right or left of the dynamic area. Only one scrollable area or dynamic area per panel can be defined with EXTEND(ON).

EXTEND(OFF) specifies that the dynamic area is not expanded. The depth of the dynamic area is defined by the box that you mark off within the)BODY section. This is the default.

In the)INIT,)REINIT, or)PROC section you can determine the number of lines in the dynamic area by using the LVLINE built-in function on an assignment statement. For more information, see "The Assignment statement" on page 90.

Note: Using EXTEND(ON) is not recommended if the panel is to be displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the screen. If the panel is then displayed in a pop-up, it might be truncated at the pop-up border.

SCROLL(ON|<u>OFF</u>)
Specifies that the dynamic area is scrollable. The scrolling commands are automatically enabled. Only one dynamic area in a panel definition can be scrollable. The default is OFF.

The value for the SCROLL keyword cannot be specified as a dialog variable.

Although the panel display service does not perform the scrolling, it does provide an interpretation of the user's scroll request.

USERMOD(*usermod-code*) DATAMOD(*datamod-code*)
Specify a character or 2-position hexadecimal value to be substituted for attribute characters in a dynamic area variable following a user interaction. The attribute characters used within the dynamic area are intermixed with the data. These attribute characters designate the beginning of a new data field within the area. When the dynamic area variable is returned to the dialog, the *usermod-code* and *datamod-code* values replace the attribute character of each field that has been modified, according to the following rules:

1. USERMOD specified, but DATAMOD not specified

   If there has been any user entry into the field, even if the field was overtyped with identical characters, the attribute byte for that field is replaced with *usermod-code*.

2. DATAMOD specified, but USERMOD not specified

   If there has been any user entry into the field, and if the value in the field has changed, either by the user entry or by SDF2/DM capitalization or justification, the attribute byte for that field is replaced with *datamod-code*.

3. Both USERMOD and DATAMOD specified

   If there has been any user entry into the field but the value in the field has not changed, the attribute byte for that field is replaced with *usermod-code*.

   If there has been any user entry into the field and the value in the field has changed, either by the user entry or by SDF2/DM capitalization or justification, the attribute byte for that field is replaced with *datamod-code*.

4. Neither DATAMOD nor USERMOD specified

   The attribute byte for the field is unchanged.

Examples:

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) USERMOD(!)
```

The character ! replaces the attribute byte for each field in the dynamic area that has been touched, not necessarily changed in value, by the user. All other attribute bytes remain as they are.

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) DATAMOD(01)
```

The hexadecimal code 01 replaces the attribute byte for each field in the dynamic area that has been touched by the user and has changed in value. All other attribute bytes remain as they are.

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) USERMOD(0C) DATAMOD(03)
```

The hexadecimal code 0C replaces the attribute byte for each field in the dynamic area that has been touched by the user, but has not changed in value. The hexadecimal code 03 replaces the attribute byte for each field in the dynamic area that has been touched by the user and has changed in value. All other attribute bytes remain as they are.

If the *datamod* or *usermod* code is one of the following special characters, it must be enclosed in single quotes in the)ATTR section:

blank < (+ |) ; ¬ − , > : =

If the required character is a single quote, use four single quotes: `DATAMOD('''')`

Panel processing considerations

When you are defining a dynamic area and generating a dynamic character string that defines the format of the data to be placed within that area on the panel, a number of rules apply:

- The area cannot be specified by using a Z-variable “place-holder” within the panel body.
- Within the dynamic area, all nonattribute characters are treated as data to be displayed. Unlike other parts of the panel body, a variable name does not follow an attribute character.
- The dialog is responsible for ensuring such requirements as the integrity of data and the validity of attribute codes for the dynamic character string.
- If the dynamic area is narrower than the screen size, the panel designer must place the appropriate attribute characters around this “box” so that the data within the area is not inadvertently affected. For example, the panel designer can place fields with SKIP attributes following the rightmost boundaries so that the cursor is properly placed to the next or continued input field within the area.
- If the dialog must know the dimensions of the dynamic area before the data is formatted, invoke the PQUERY dialog service. For more information, see “PQUERY — Obtain panel information” on page 152.
- The scroll amount field is optional. On a panel with a scrollable area, if the input field following the command field in the panel body is exactly 4 characters long, it is assumed to be the scroll amount field. Otherwise, the system variable ZSCROLLD, which can be set by the dialog, is used to determine the default scroll amount. If there is no scroll amount field and ZSCROLLD has not been set, PAGE is assumed.

ZSCROLLA contains the value of the scroll amount field, such as MAX or CSR. ZSCROLLN contains the scroll number computed from the value in the scroll amount field (number of lines or columns to scroll). For example, if the dynamic area occupies 12 lines on a 22-line terminal and the user requests DOWN HALF, ZSCROLLN contains a 6. The system variable ZVERB contains the scroll direction, DOWN in this case. If ZSCROLLA has a value of MAX, the value of ZSCROLLN is not meaningful.

Although the panel display service does not perform the scrolling for dynamic areas, it does provide an interpretation of the user's scroll request.

Within a panel definition, you can specify at most one scrollable area or dynamic area with EXTEND(ON). You can also specify at most one dynamic area that is scrollable. Figure 38 shows a panel definition containing one dynamic area that is defined with EXTEND(ON) and is scrollable.

```
)ATTR
 #   AREA(DYNAMIC)  SCROLL(ON)  EXTEND(ON)
)BODY
%------------------- TITLE ----------------------
%COMMAND ===>_ZCMD                 +SCROLL ===>_AMT +
+
+  (Instructions for this panel ...)
+
#DAREA ------------------------------------------#
+
+  (More instructions for this panel ...)
+
```

Figure 38. Panel definition illustrating SCROLL and EXTEND

In the example in Figure 38, there are:

- Five lines in the panel body before the extended area
- Three more lines after the extended area

This makes a total of eight lines that are outside the dynamic area. Therefore, if the panel were displayed on an IBM 3278 Model 4 display device, which has 43 lines, the depth or extent of the dynamic area would be 43 minus 8, or 35 lines.

In this example, the dynamically generated data string to be placed in the area is taken from the dialog variable SAREA. If, for example, the dynamic area is 60 characters wide and 10 lines deep, the first 60 characters of the string are placed in the first line of the area, the next 60 characters are placed in the second line of the area, and so on, until the last 60 characters are placed in the tenth line of the area. Following a user interaction, the contents of the area are stored in the same variable.

The width of the dynamic area includes the special characters that designate the vertical sides. These delimiter characters do not represent attribute characters.

A number of the capabilities described in the previous sections have implications for both panel areas and panel fields. These include the following:

- The cursor placement capability applies to dynamic areas. That is, .CURSOR can be assigned to a dynamic area name and .CSRPOS can be assigned to a position within the dynamic area. The position within an area applies within the rectangular bounds of that area.
- The .ATTRCHAR control variable can be used to override attribute characters that are used within dynamic areas. In addition, .ATTRCHAR can be used to define a new attribute character that has not been previously listed within the panel)ATTR section. Using .ATTRCHAR as a vehicle for defining new attribute characters can be done only within the)INIT section and only for fields within dynamic areas (type DATAIN, DATAOUT, or DATAOUTGE).
- The PQUERY service can be invoked by the application prototype to determine the characteristics of the dynamic area before the application prototype constructs the dynamic character string.

Help for a dynamic area

To display help for a field within a dynamic area, use the HELP dialog management service. For more information, see “The HELP service” on page 148.

Example of a panel body

Figure 39 on page 68 shows a simple panel definition that uses the default attribute characters.

This data entry panel has 11 input fields indicated with the _ attribute character. It also has a substitutable variable (EMPSER) within a text field.

The 1-character input field for INITIAL and the 3-character input field for AREA CODE are not long enough to contain their dialog variables. In each case, the placeholder variable Z is coded. The .ZVARS statement in the)INIT section says that the first Z is to be replaced by MINIT and the second Z is to be replaced by PHAC.

The first two lines of the panel and the arrows preceding the input fields are all highlighted, as indicated by the % attribute character. The other text fields are low intensity, as indicated by the + attribute character.

```
)BODY
%----------------------------  EMPLOYEE RECORDS  ------------------------------
%COMMAND ===>_ZCMD                                                            %
%
%EMPLOYEE SERIAL: &EMPSER
+
+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+
+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_Z+
+
+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                 +
+     LINE 2 %===>_ADDR2                                 +
+     LINE 3 %===>_ADDR3                                 +
+     LINE 4 %===>_ADDR4                                 +
+
+   HOME PHONE:
+     AREA CODE   %===>_Z  +
+     LOCAL NUMBER%===>_PHNUM   +
+
)INIT
.ZVARS = '(MINIT, PHAC)'
)END
```

Figure 39. Panel body example

Figure 40 shows the panel as it appears when displayed, assuming that the current value of EMPSER is 123456 and the other variables are initially null.

```
----------------------------  EMPLOYEE RECORDS  ------------------------------
COMMAND ===>

EMPLOYEE SERIAL: 123456

   TYPE OF CHANGE ===>            (NEW, UPDATE, OR DELETE)

   EMPLOYEE NAME:
     LAST    ===>
     FIRST   ===>
     INITIAL ===>

   HOME ADDRESS:
     LINE 1  ===>
     LINE 2  ===>
     LINE 3  ===>
     LINE 4  ===>

   HOME PHONE:
     AREA CODE    ===>
     LOCAL NUMBER ===>
```

Figure 40. Panel body example, when displayed

The)AREA section

The)AREA section is optional. It lets you define scrollable areas on a panel. A user can see and interact with the total content defined for the scrollable area by scrolling.

If a panel contains more than one scrollable area, code a separate)AREA section for each scrollable area.

For each scrollable area, do the following:

- In the)ATTR section, specify an attribute character that defines the borders of the scrollable area. You also specify whether the scrollable area is to expand to fill all the available space on the screen.
- In the)BODY section, use the attribute character to mark the place in the panel where the scrollable area is to be displayed.
- In the)AREA section, put the contents of the scrollable area.

Specifying the area attribute in the)ATTR section

In the)ATTR section, specify an attribute character. You can use the same attribute character for several scrollable areas if the scrollable areas all have the value EXTEND(OFF).

The syntax is as follows:

Syntax

attribute-char **AREA(SCRL)** [**EXTEND(ON|OFF)**]

attribute-char
Attribute character. This attribute defines the borders of the scrollable area in the)BODY section.

AREA(SCRL)
Indicates that this attribute defines the location of a scrollable area in the)BODY section.

EXTEND(ON|OFF)
EXTEND(ON) specifies that the depth of the scrollable area is to be automatically increased so that the panel fills the physical screen on which it is displayed. In the)BODY section, you need provide only the first line of the scrollable area, unless other text or fields are to be displayed to the right or left of the scrollable area. For each panel, you can define only one scrollable area or dynamic area with EXTEND(ON).

EXTEND(OFF) specifies that the scrollable area is not to be expanded. The depth of the scrollable area is defined by the box that you mark off within the)BODY section. This is the default.

In the)INIT, REINIT, or)PROC section you can determine the number of lines in the scrollable area by using the LVLINE built-in function on an assignment statement. For more information, see “The Assignment statement” on page 90.

Note: Using EXTEND(ON) is not recommended if the panel will be displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the screen. If the panel is then displayed in a pop-up, the panel might be truncated at the pop-up border.

Mapping the area in the)BODY section

In the)BODY section, you specify the appearance of the panel. For a scrollable area, you specify the top, left, and right borders. Immediately after the attribute character on the first line of the scrollable area, you specify the name of the scrollable area.

If you accept the default of EXTEND(OFF), you indicate the depth of the scrollable area by the number of times you repeat the attribute character within the)BODY section. For example, your)ATTR and)BODY sections might look like this:

```
)ATTR
  # AREA(SCRL)
)BODY
#myarea---------#
#               #
#               #
#               #
```

If you specify EXTEND(ON) in the)ATTR section, the scrollable area is automatically expanded so the panel fills the physical screen. In this case, you need specify only the first line of the scrollable area in the)BODY section. For example:

```
)ATTR
  # AREA(SCRL) EXTEND(ON)
)BODY
#myarea---------#
```

The width of the scrollable area includes the special characters that designate the vertical sides. (In the previous examples, these are the # characters.)

Special considerations for scrollable areas

When you are marking a scrollable area in the)BODY section, a number of rules apply:

- The area cannot be specified by using a Z-variable placeholder within the panel body.
- To allow for the scroll information, the minimum width for a scrollable area is 20.
- If the width of the scrollable area is less than the screen size, place appropriate attribute characters around this area so that the data within the area is not inadvertently affected. For example, by using place fields with SKIP attributes following the rightmost boundaries of the area, you can ensure that the cursor will tab correctly to the next or continued input field within the area.
- Text fields or fields in the scrollable area cannot be defined to wrap. A field cannot extend beyond one line of the area.

If EXTEND(ON) has been specified, the last line of the scrollable area definition is repeated for each line that the panel is expanded. Therefore, only text can appear on the last line of the area definition.

It is good practice to frame a scrollable area or to allow enough blank space so that the definition of the scrollable area is clear. Consult your own usability standards to determine the best implementation.

The)AREA header statement

The)AREA section begins with the)AREA header statement. It has the following syntax:

Syntax

```
)AREA name[DEPTH(depth)]
          [WIDTH(width)]
          [SCRLTEXT(indicator-text)]
          [TOPROW(variable-name)]
          [LEFTCOL(variable-name)]
          [CONTCHAR(c)]
```

name
Specifies the name of the scrollable area that is to be matched with the name specified in the)BODY section. This name cannot be specified as a dialog variable.

DEPTH(*depth***)**
Specifies the minimum number of lines in the scrollable area (not including the scroll indicator line) when EXTEND(ON) has been specified. DEPTH has no effect when EXTEND(OFF) is used. The top line is always reserved for the scroll information and is not considered part of the depth value.

DEPTH can be used to ensure that a required number of lines are displayed. The depth value cannot be specified as a dialog variable. It must be greater than or equal to the number of lines defined for the area in the)BODY section and less than or equal to the number of lines in the)AREA definition.

WIDTH(*width***)**
To allow horizontal scrolling, specify a width value greater than the area width in the panel body. The maximum value is 255. Do not use this in help panels, because the LEFT and RIGHT commands have a different meaning in help.

SCRLTEXT(*indicator-text***)**
Specifies the text used for the scroll indicator. The default is `'More:'`

TOPROW(*variable-name***)**
Holds the line number of the area data displayed as the top line. Set during scrolling, it is used for the panel display. The valid range is 1–65534. A TOPROW value of zero is ignored. If the TOPROW value is greater than the area size the last line is displayed as the top line.

LEFTCOL(*variable-name***)**
Holds the column number of the scrollable area data displayed as the leftmost column. Set during scrolling, it is used for the panel display. The valid range is 1 to the value of *width*. A LEFTCOL value of zero is ignored. If the LEFTCOL value is greater than the width of the scrollable area the last column is displayed as the leftmost column.

CONTCHAR(*single-character*)
Specifies a single character, which is used as a continuation character in the)AREA section. The character must be the last character of the record. Any character except X'00' is allowed. The next record is then concatenated with current one starting at the position of the continuation character.

Statements within the)AREA section

After the)AREA header statement, put the text and fields that make up the scrollable area. The)AREA section can contain anything that the)BODY section can contain except:

The command field
The short message field
The long message field
The action bar
Dynamic areas
Other scrollable areas
Scrollable lists

How the panel is displayed

The cursor position determines how an area scrolls. If scroll down is requested and the cursor is currently on the top line of the scrollable area, the section is scrolled so that the last visible line becomes the top line. Otherwise, the line containing the cursor is moved to the top line.

The top line of the scrollable area is reserved for the scroll indicators. Actual information from the)AREA section is displayed beginning on the second line of the scrollable area.

Scroll indicators are displayed if more data was defined in the)AREA section than fits in the panel area. The following scroll indicators can be displayed:

+	You can scroll forward.
−	You can scroll backward.
<	You can scroll left.
>	You can scroll right.

These scroll indicators can be combined. For example, if you can scroll either left or right but not forward or backward, the scroll indicator will look like this:

```
More: < >
```

Define forward and backward PF keys (corresponding to the commands DOWN and UP) in the keylist for any application prototype panel that has scrollable areas.

Scrollable area within a help panel

A scrollable area within a help panel is displayed in the same way as any other scrollable area, but the scroll commands are different. When a help panel contains a scrollable area, the LEFT command scrolls up, and the RIGHT command (or just pressing Enter) scrolls down.

If the help panel has more than one scrollable area, the cursor position determines which scrollable area is scrolled. If the cursor is not within any of the scrollable areas, the first scrollable area is scrolled.

In the keylist that you use for help panels, you may want to assign function keys for LEFT and RIGHT.

How the panel is processed

When a DISPLAY service is called, the)INIT section is processed before the panel is displayed. The)REINIT and)PROC sections are not processed every time you scroll. The PROC section is processed only when the panel is submitted for processing, for example by pressing the Enter key.

When panel processing is complete and SDF2/DM returns control to the dialog, it is possible that required fields are not displayed. To prevent this, use the VER statement as described in "The VER statement" on page 97.

When fields are displayed on a panel, their characteristics can change without the user interacting with the fields. For example, when CAPS(ON) is set for a field, this affects only fields that are actually displayed. If a field is initialized with lowercase letters and appears on a portion of the panel that is never displayed, the data remains in lowercase even if CAPS(ON) was set for the field.

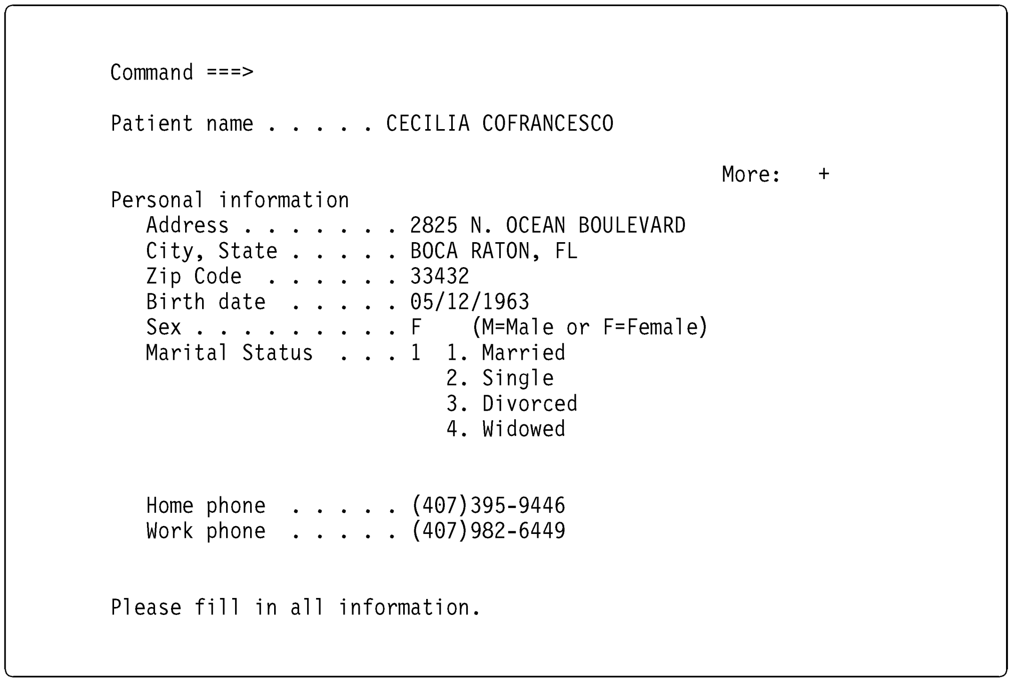
Example

Figure 41 shows a scrollable area definition. It is followed by the actual scrollable panel displays.

```
)ATTR
  # AREA(SCRL) EXTEND(ON)
)BODY
%
%Command ===>_ZCMD
%
+Patient name . . . . ._pname                       %
+
#myarea -----------------------------------------------------------#
+
+Please fill in all information.
+
)AREA MYAREA DEPTH(5)
+Personal information
+   Address . . . . . . ._address                     %
+   City, State . . . . ._ctyst                       %
+   Zip Code  . . . . . ._zip  %
+   Birth date  . . . . ._birth     %
+   Sex . . . . . . . . ._SX%  (M=Male or F=Female)
+   Marital Status  . . ._MS+1. Married
+                            2. Single
+                            3. Divorced
+                            4. Widowed
+
+
+   Home phone  . . . . ._hphone       %
+   Work phone  . . . . ._wphone       %
+
+Emergency Contact
+   Name  . . . . . . . ._ename                        %
+   Home phone  . . . . ._ehphone      %
+   Work phone  . . . . ._ewphone      %
+
+Insurance Coverage
+   Insurance Company . ._insure                        %
+   Group number  . . . ._gn%
+   ID number . . . . . ._ID   %
+   Cardholder's name . ._cname                         %
+   Relationship  . . . ._RL+1. Self
+                           +2. Spouse
+                           +3. Parent
+                           +4. Relative
+                           +5. Other
+  Signature on file  . ._SG+ (Y=Yes N=No)
)INIT
  ⋮
)PROC
  ⋮
)HELP
  ⋮
)END
```

Figure 41. Scrollable area definition

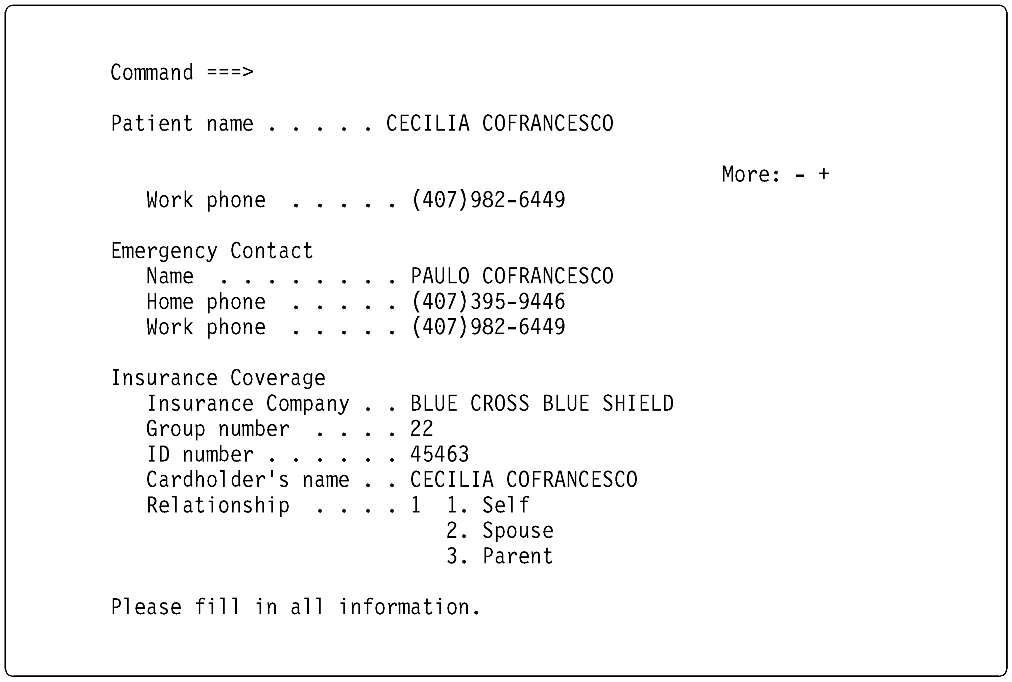
Figure 42 shows the initial panel display, which contains a scrollable area. `More:        +` indicates that you can now scroll forward in the scrollable area.



```
Command ===>

Patient name . . . . . CECILIA COFRANCESCO

                                                   More:   +
Personal information
   Address . . . . . . . 2825 N. OCEAN BOULEVARD
   City, State . . . . . BOCA RATON, FL
   Zip Code  . . . . . . 33432
   Birth date  . . . . . 05/12/1963
   Sex . . . . . . . . . F    (M=Male or F=Female)
   Marital Status  . . . 1  1. Married
                            2. Single
                            3. Divorced
                            4. Widowed


   Home phone  . . . . . (407)395-9446
   Work phone  . . . . . (407)982-6449


Please fill in all information.
```

Figure 42. Scrollable area screen display (Part 1 of 3)

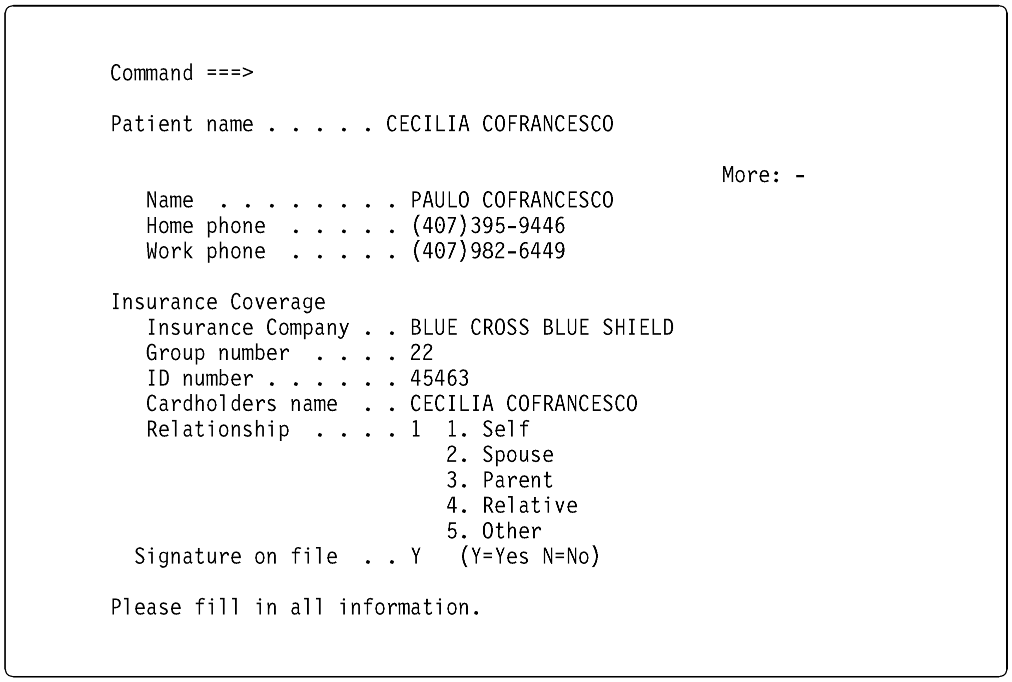
Figure 43 shows the panel display after one scroll request has been processed. `More:   - +` indicates that you can now scroll forward or backward in the scrollable area.



```
Command ===>

Patient name . . . . . CECILIA COFRANCESCO

                                                   More: - +
   Work phone  . . . . . (407)982-6449

Emergency Contact
   Name  . . . . . . . . PAULO COFRANCESCO
   Home phone  . . . . . (407)395-9446
   Work phone  . . . . . (407)982-6449

Insurance Coverage
   Insurance Company . . BLUE CROSS BLUE SHIELD
   Group number  . . . . 22
   ID number . . . . . . 45463
   Cardholder's name . . CECILIA COFRANCESCO
   Relationship  . . . . 1  1. Self
                            2. Spouse
                            3. Parent

Please fill in all information.
```

Figure 43. Scrollable area screen display (Part 2 of 3)

Figure 44 shows the panel display after you have completely scrolled through the scrollable area. `More:    -` indicates that you can now only scroll backward in the scrollable area.



```
Command ===>

Patient name . . . . . CECILIA COFRANCESCO

                                                 More: -
   Name  . . . . . . . . PAULO COFRANCESCO
   Home phone  . . . . . (407)395-9446
   Work phone  . . . . . (407)982-6449

Insurance Coverage
   Insurance Company . . BLUE CROSS BLUE SHIELD
   Group number  . . . . 22
   ID number . . . . . . 45463
   Cardholders name  . . CECILIA COFRANCESCO
   Relationship  . . . . 1  1. Self
                            2. Spouse
                            3. Parent
                            4. Relative
                            5. Other
  Signature on file  . . Y   (Y=Yes N=No)

Please fill in all information.
```

Figure 44. Scrollable area screen display (Part 3 of 3)

The)LIST section

The)LIST section is optional. You use it to define scrollable lists on a panel. A scrollable list is similar to a scrollable area. It usually consists of some lines representing header information, followed by a number of lines with identical format. Each of these lines contains one or more fields, which are elements of an array. Only the first of these lines has to be specified. This line is automatically repeated for each index of the index range up to the end of the scrollable list. You can define the index range in the NUMROWS parameter of the)LIST statement. If no NUMROWS value is specified, it is determined by the dimension of the first variable in the line. If it is a REXX variable, it is defined by the *variable*.0 value; otherwise, it is defined by the dimension parameter of the VDEFINE service. A user can see and interact with the total content defined for the scrollable list by scrolling.

If a panel contains more than one scrollable list, code a separate)LIST section for each scrollable list.

For each scrollable list, do the following:

- In the)ATTR section, specify an attribute character that identifies the borders of the scrollable list. Specify also whether the scrollable list is to fill all the available space on the screen.
- In the)BODY section, use the attribute character to mark the place in the panel where the scrollable list will be displayed.
- In the)LIST section, put the contents of the scrollable list.

Specifying the attribute for the scrollable list in the)ATTR section

In the)ATTR section, specify an attribute character. You can use the same attribute character for several scrollable lists if all the scrollable lists have the value EXTEND(OFF).

The syntax is as follows:

Syntax

attribute-char **AREA(LIST)** [**EXTEND(ON|<u>OFF</u>)**]

attribute-char
Attribute character. This attribute defines the borders of the scrollable list in the)BODY section.

AREA(LIST)
Indicates that this attribute defines the location of a scrollable list in the)BODY section.

EXTEND(ON|<u>OFF</u>)
EXTEND(ON) specifies that the depth of the scrollable list is to be automatically increased so that panel fills the physical screen on which it is displayed. In the)BODY section, you need provide only the first line of the scrollable list, unless other text or fields are to be displayed to the right or left of the scrollable list. For each panel, you can define only one scrollable list or dynamic area with EXTEND(ON).

EXTEND(OFF) specifies that the scrollable list is not to be expanded. The depth of the scrollable list is defined by the box that you mark off within the)BODY section. This is the default.

In the)INIT,)REINIT, or)PROC section, you can determine the number of lines in the scrollable list by using the LVLINE built-in function on an assignment statement. For more information, see "The Assignment statement" on page 90.

Note: Using EXTEND(ON) is not recommended if the panel is to be displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the screen. If the panel is then displayed in a pop-up, it might be truncated at the pop-up border.

Mapping the scrollable list in the)BODY section

In the)BODY section, specify the appearance of the panel. For a scrollable list, specify the top, left, and right borders. Immediately after the attribute character on the first line of the scrollable list, specify the name of the scrollable list.

If you accept the default of EXTEND(OFF), indicate the depth of the scrollable list by the number of times you repeat the attribute character within the)BODY section. For example, your)ATTR and)BODY sections might look like this:

```
)ATTR
  # AREA(LIST)
)BODY
#mylist---------#
#               #
#               #
#               #
```

If you specify EXTEND(ON) in the)ATTR section, the scrollable list is automatically expanded so the panel fills the physical screen. In this case, you need specify only the first line of the scrollable list in the)BODY section. For example:

```
)ATTR
  # AREA(LIST) EXTEND(ON)
)BODY
#mylist---------#
```

The width of the scrollable list includes the special characters that designate the vertical sides. (In the preceding examples, these are the # characters.)

Special considerations for scrollable lists

When you are marking a scrollable list in the)BODY section, the following restrictions apply:

- The list cannot be specified by using a Z-variable placeholder within the panel body.
- To allow for the scroll information, the minimum width for a scrollable list is 20.
- If the width of the scrollable list is less than the screen size, place appropriate attribute characters around this list so that the data within the list is not inadvertently affected. For example, by using place fields with SKIP attributes following the rightmost boundaries of the list, you can ensure that the cursor will tab correctly to the next or continued input field within the list.
- Text fields or fields in the scrollable list cannot be defined to wrap. A field cannot extend beyond one line of the list.

If EXTEND(ON) has been specified, the last line of the scrollable list definition is repeated for each line that the panel is expanded.

It is good practice to frame a scrollable list or to allow enough blank space so that the definition of the list is clear. Consult your own usability standards to determine the best implementation.

The)LIST header statement

The)LIST section begins with the)LIST header statement. The)LIST statement and all associated keywords must be specified on the same line. It has the following syntax:

Syntax

)LIST *name* [**DEPTH(***depth***)**]
[**WIDTH(***width***)**]
[**SCRLTEXT(***indicator-text***)**]
[**TOPROW(***variable-name***)**]
[**NUMROWS(***variable-name***)**]
[**MODVAR(***variable-name***)**]
[**LEFTCOL(***variable-name***)**]
[**CONTCHAR(***c***)**]

or

)LIST *name* [**WIDTH(***width***)**]
[**LEFTCOL(***variable-name***)**]
[**PARENT(***list-name***)**]
[**CONTCHAR(***c***)**]

name
Specifies the name of the scrollable list that is to be matched with the name specified in the)BODY section. This name cannot be specified as a dialog variable.

DEPTH(*depth***)**
Specifies the minimum number of lines in the scrollable list (not including the scroll indicator line) when EXTEND(ON) has been specified. DEPTH has no effect when EXTEND(OFF) is used. The top line is always reserved for the scroll information and is not considered part of the depth value.

DEPTH can be used to ensure that a required number of lines are displayed. The depth value cannot be specified as a dialog variable. It must be greater than or equal to the number of lines defined for the list in the)BODY section and greater than or equal to the number of lines in the)LIST definition.

WIDTH(*width***)**
To provide horizontal scrolling, specify a width value greater than the list width in the panel body. The maximum value is 255. Do not use this in help panels, because the LEFT and RIGHT commands have a different meaning in help.

SCRLTEXT(*indicator-text***)**
Specifies the text used for the scroll indicator. The default is `More:`.

TOPROW(*variable-name***)**
Holds the line number of the scrollable list data displayed as the top line. Set during scrolling, it is used for the panel display. This number does not include any optional header lines of the list. The valid range is 1–65534. A TOPROW value of zero is ignored. If the TOPROW value is greater than the dimension of the scrollable list, the last line is displayed as the top line.

LEFTCOL(*variable-name***)**
Holds the column number of the scrollable list data displayed as the leftmost column. It is used for the panel display. The LEFTCOL value is subsequently set automatically when the user scrolls the panel. The valid range is 1 to *width*. A LEFTCOL value of zero is ignored. If the LEFTCOL value is greater than the WIDTH value of the scrollable list, the last column is displayed as the leftmost column.

NUMROWS(*variable-name***)**
This variable holds the dimension of the scrollable list. This number does not include any optional header lines of the list. The valid range is 1–65534. Do not use values greater than the dimension of a variable in the list.

MODVAR(*variable-name***)**
This keyword specifies the name of the array variable that indicates which rows in the list field have been modified.

Normally, an application prototype will initialize the first element of the array to 0 and then call the DISPLAY service. After returning from the DISPLAY service call, the nonzero elements before the first 0 entry each contain the row number of a modified row (the first row is row number 1). The sequence of modified-row numbers ends with the first 0 element or the end of the array, whichever occurs first.

In a REXX procedure, the variable name is considered a stem. The *variable*.0 value has to be initialized to 0 before the display service is called. It is set by the DISPLAY service if lines have been modified. The *variable*.0 value indicates how many lines have been changed. The *variable.n* values (*n*=1 to *variable*.0) are the line numbers of the modified lines.

PARENT(*list-name***)**
The PARENT keyword defines this list as a child of the list specified by the *list-name* value. This list is therefore automatically scrolled vertically, together with the parent list. The keyword values for DEPTH, TOPROW, NUMROWS, and MODVAR—if you specify them—are identical to those of the parent list. Because the parent list also holds the SCRLTEXT definition, you should position the parent list as the rightmost list.

CONTCHAR(*single-character***)**
Specifies a single character, which is used as a continuation character in the)LIST section. The character must be the last character of the record. Any character except `X'00'` is allowed. The next record is then concatenated with the current record, starting at the position of the continuation character.

Statements within the)LIST section

After the)LIST header statement, put the text and fields that make up the scrollable list. All lines, except the last line in the definition, are treated as header lines and are always displayed. The last line is used as a model line. The)LIST section can contain anything that the)BODY section can contain except:

The command field
The short message field
The long message field
The action bar
Dynamic areas
Scrollable areas
Other scrollable lists

How the panel is displayed

Cursor position determines how a list scrolls. If scroll down is requested and the cursor is currently on the top line of the scrollable area, the section is scrolled so that the last visible line becomes the top line. Otherwise, the line containing the cursor is moved to the top line.

The top line of the scrollable list is reserved for the scroll indicators. Actual information from the)LIST section is displayed beginning on the second line of the scrollable list.

Scroll indicators are displayed if more data was defined in the)LIST section than fits in the panel list. The following scroll indicators can be displayed:

+	Scroll forward.
–	Scroll backward.
<	Scroll left.
>	Scroll right.

These scroll indicators can be combined. For example, if you can scroll either left or right but not forward or backward, the scroll indicator will look like this:

```
More: < >
```

Define forward and backward program function keys (corresponding to the commands DOWN and UP) in the keylist for any application prototype panel that has scrollable lists.

Scrollable list within a help panel

A scrollable list within a help panel is displayed in the same way as any other scrollable list, but the scroll commands are different. When a help panel contains a scrollable list, the LEFT command scrolls up, and the RIGHT command (or just pressing Enter) scrolls down.

If the help panel has more than one scrollable list, the cursor position determines which scrollable list is scrolled. If the cursor is not within any of the scrollable lists, the first scrollable list is scrolled.

In the keylist that you use for help panels, you may want to assign function keys for LEFT and RIGHT.

How the panel is processed

When a DISPLAY service is called, the)INIT section is processed before the panel is displayed. The)REINIT and)PROC sections are not processed every time you scroll. The)PROC section is processed only when the panel is submitted for processing, for example by pressing the Enter key.

When panel processing is complete and SDF2/DM returns control to the dialog, it is possible that required fields are not displayed. To prevent this, use the VER statement, as described in “The VER statement” on page 97.

When fields are displayed on a panel, their characteristics can change without the user interacting with the fields. For example, when CAPS(ON) is set for a field, this affects only fields that are actually displayed. If a field is initialized with lowercase letters and appears on a portion of the panel that is never displayed, the data remains in lowercase even if CAPS(ON) was set for the field.

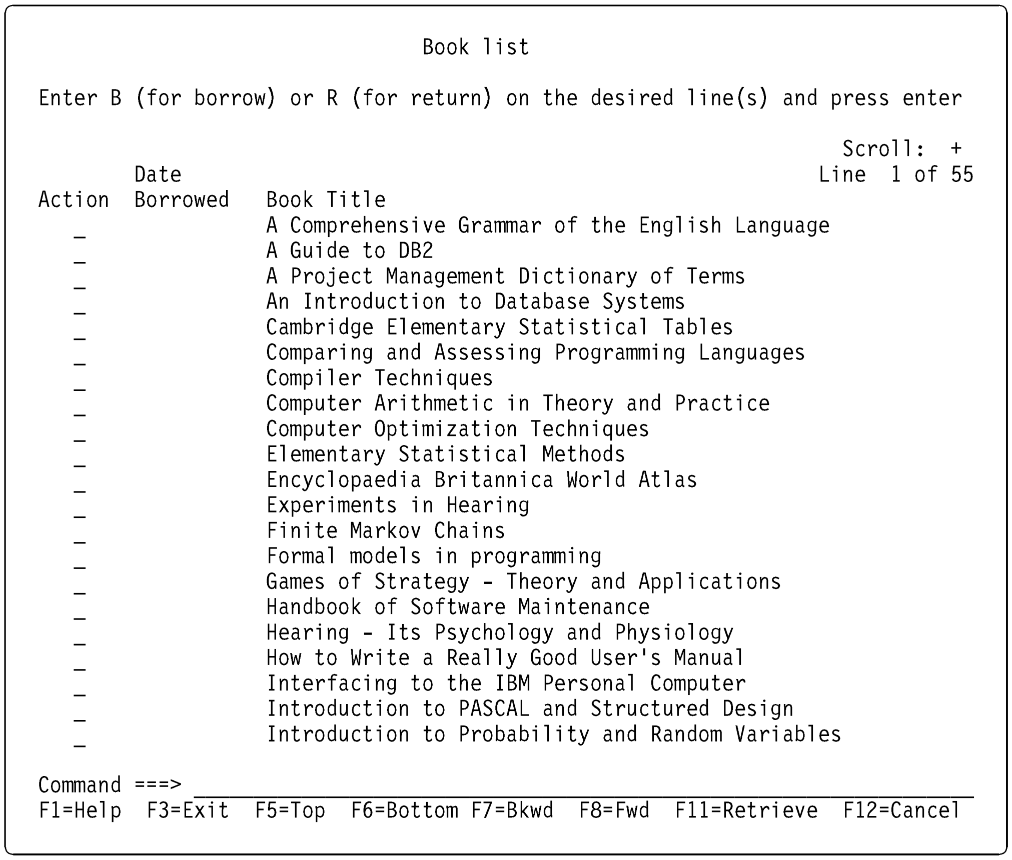
Example

Figure 45 shows a scrollable list definition. It is followed by the actual scrollable panel display.

```
)PANEL
)ATTR
  { TYPE(PT)                      /* panel title                 */
  [ TYPE(MSGS)                    /* short message               */
  \ TYPE(PIN)                     /* panel instructions          */
  ? TYPE(TEXT) COLOR(WHITE)
  ! TYPE(LI)                      /* list item                   */
  _ TYPE(LEF) CAPS(ON)            /* list entry field            */
  @ TYPE(CH)                      /* column header               */
  " TYPE(TEXT) CUATYPE(MSGS)
  # TYPE(OUTPUT) CUATYPE(MSGS) JUST(RIGHT)
  $ AREA(LIST) EXTEND(ON)
)BODY SMSG(DGISMSG)
+
                                {Book list           [DGISMSG
+
\Enter%B\(for borrow) or%R\(for return) on the desired line(s) and press enter
+
$list1 ----------------------------------------------------------------------$
%&DGILMSG
%Command ===>_ZCMD                                                           +
)LIST LIST1 SCRLTEXT('Scroll:') TOPROW(TOPL)
       @Date                                                   "Line#Z "of#Z +
@Action  Borrowed   Book Title
   _a+!bdate       !title
)INIT
 .ZVARS='(TOPL NUML)'
 .ALARM = &DGIALARM
 &ZCMD=''
)REINIT
)PROC
 VER(&A,LIST,R,B)
 &LSTIDX=.CSRIDX
 &LASTLN=LVLINE(LIST1)
)KEYLIST
 key(f1)  cmd(help)     fka(yes) text('Help ')
 key(f3)  cmd(exit)     fka(yes) text('Exit ')
 key(f5)  cmd(top)      fka(yes) text('Top ')
 key(f6)  cmd(bottom)   fka(yes) text('Bottom')
 key(f7)  cmd(up)       fka(yes) text('Bkwd ')
 key(f8)  cmd(down)     fka(yes) text('Fwd ')
 key(f11) cmd(retrieve) fka(yes) text('Retrieve ')
 key(f12) cmd(cancel)   fka(yes) text('Cancel')
)END
```

Figure 45. Scrollable list definition

Figure 46 on page 84 shows the initial panel display, which contains a scrollable list. `Scroll:`   + indicates that you can now scroll forward in the scrollable list.



```
                                Book list

Enter B (for borrow) or R (for return) on the desired line(s) and press enter

                                                                 Scroll:  +
        Date                                                   Line  1 of 55
Action  Borrowed   Book Title
   _               A Comprehensive Grammar of the English Language
   _               A Guide to DB2
   _               A Project Management Dictionary of Terms
   _               An Introduction to Database Systems
   _               Cambridge Elementary Statistical Tables
   _               Comparing and Assessing Programming Languages
   _               Compiler Techniques
   _               Computer Arithmetic in Theory and Practice
   _               Computer Optimization Techniques
   _               Elementary Statistical Methods
   _               Encyclopaedia Britannica World Atlas
   _               Experiments in Hearing
   _               Finite Markov Chains
   _               Formal models in programming
   _               Games of Strategy - Theory and Applications
   _               Handbook of Software Maintenance
   _               Hearing - Its Psychology and Physiology
   _               How to Write a Really Good User's Manual
   _               Interfacing to the IBM Personal Computer
   _               Introduction to PASCAL and Structured Design
   _               Introduction to Probability and Random Variables

Command ===> _________________________________________________________________
F1=Help  F3=Exit  F5=Top  F6=Bottom F7=Bkwd  F8=Fwd  F11=Retrieve  F12=Cancel
```

Figure 46. Scrollable list screen display

The)INIT section

The)INIT (initialization) section is optional. It specifies the initial processing that occurs before the panel is displayed.

The)INIT section begins with an)INIT header statement and ends with a)REINIT,)PROC,)HELP,)KEYLIST, or)END header statement. The)INIT header statement has no keywords. The number of lines allowed in an)INIT section depends on the storage size available for panel processing at execution time.

The)INIT,)REINIT, and)PROC sections can contain the following types of statements:

Assignment
ELSE
EXIT
GOTO
IF
REFRESH —)REINIT and)PROC sections only
VER (verify)
VGET
VPUT

These statements are described in detail in Chapter 3, “Panel processing statements” on page 89.

These statements can use the following control variables:

.ALARM
.ATTR
.ATTRCHAR
.CSRIDX
.CSRPOS
.CURSOR
.HELP
.MSG
.RESP
.TRAIL
.ZVARS —)INIT section only

The assignment statement can use the following built-in functions:

LVLINE (last visible line)
TRANS (translate)
TRUNC (truncate)

The control variables are described in detail in Chapter 4, “Control variables” on page 100. The built-in functions are described in detail in “The Assignment statement” on page 90.

After the)INIT section is processed, the variables in the panel body are replaced with the corresponding values (with trailing blanks stripped) and the panel is displayed.

The)REINIT section

The)REINIT (re-initialization) section is optional. It specifies the processing that occurs when the panel is redisplayed.

The)REINIT section begins with a)REINIT header statement and ends with a)PROC,)HELP,)KEYLIST, or)END header statement. The)REINIT header statement has no keywords. The number of lines allowed in a)REINIT section depends on the storage size available for panel processing at execution time.

The)REINIT section can contain the same types of statements, control variables, and built-in functions as can the)INIT section. For more information, see Chapter 3, “Panel processing statements” on page 89 and Chapter 4, “Control variables” on page 100.

A panel is redisplayed in either of the following situations:

- After the)PROC section has been processed, if the .MSG control variable is nonblank and the user has not requested CANCEL, END, EXIT, or RETURN. The .MSG control variable is set automatically if a translation or verification error occurs. It can also be set explicitly by use of an assignment statement in the)PROC section.
- If an application prototype invokes the DISPLAY service with no panel name.

When a panel is redisplayed, the)INIT section is not processed; the)REINIT section is processed instead. The automatic fetching of variables in the panel body is also bypassed. The panel is redisplayed exactly as the user last saw it, except:

- An error message can appear.
- Field attribute overrides, assignment statements, or REFRESH statements can be used.
- A scrollable area can be scrolled to position the cursor or to verify failure.

Typically, a)REINIT section contains:

- Field attribute overrides, specified by the .ATTR control variable
- Changes to displayed panel fields, specified by assignment statements and the REFRESH statement

The)PROC section

The)PROC (processing) section is optional. It specifies the processing that occurs when the user has finished interacting with the panel.

The)PROC section begins with a)PROC header statement and ends with a)HELP,)KEYLIST, or)END header statement. The)PROC header statement has no keywords. The number of lines allowed in a)PROC section depends on the storage size available for panel processing at execution time.

The)PROC section can contain the same types of statements, control variables, and built-in functions as can the)INIT section. For more information, see Chapter 3, “Panel processing statements” on page 89 and Chapter 4, “Control variables” on page 100.

A statement can be continued over as many lines as necessary, provided that it is broken at the end of a word, or that a continuation symbol (+) is used within a literal.

Before the)PROC section is processed, input fields are automatically stored in the corresponding dialog variables.

The)HELP section

The)HELP section is optional. It specifies which help panel, if any, is to be displayed when help is requested for a panel element. Help can be requested for a field, action-bar choice, pull-down choice, or reference phrase.

The)HELP section begins with the)HELP header statement, which has no parameters.

Within the)HELP section you code statements with the following syntax:

Syntax

FIELD(*field-name***)** {**PANEL(***help-panel-name***)**|**APPL**}

FIELD(*field-name***)**
The name of the panel element for which a help panel is to be displayed.

The *field-name* must have the following characteristics:

- It must be 1–8 characters in length.
- The first character must be A–Z or a–z.
- Any remaining characters must be A–Z, a–z, or 0–9.

For an action-bar choice or pull-down choice, the name is the value that you specified in the NAME keyword of the)ABC or PDC statement, as described in "The)ABC section" on page 51. For a field, the name is the variable name. For a reference phrase, the name takes the form ZRP*xxyyy*, where:

xx	Is 00 if the reference phrase appears in the)BODY section, or a number from 01 through 99 if it appears in an)AREA section. The area sections are numbered, based on where they appear within the panel body. Numbering starts at the top left-hand corner of the panel and continues through each line of the panel from left to right.
yyy	Is the number of the reference phrase within its)BODY or)AREA section, from 001 through 999. Numbering starts at the top left-hand corner of the)BODY or)AREA section and continues through each line of the section from left to right.

PANEL(*help-panel-name***)**
The name of the help panel associated with the field.

The *help-panel-name* must have the following characteristics:

- It must be 1–8 characters in length.
- The first character must be A–Z or a–z.
- Any remaining characters must be A–Z, a–z, or 0–9.

You can specify a variable name, preceded by an ampersand (&). The value of the variable is used as the name of the help panel.

APPL
Returns the help command to the application prototype. When help is requested for this field, ZVERB is set to HELP and SDF2/DM returns to the application prototype with return code 0.

The)KEYLIST section

The)KEYLIST section is optional. A keylist determines what happens when a user presses a function key while a panel is being displayed.

You can either include a keylist in the)KEYLIST section, or make the keylist a separate file and refer to the file on the)PANEL statement. If the keylist is a separate file, it can be used by several panels.

For more information about keylists, see Chapter 5, “Keylists” on page 107.

The)END section

The)END section is required. It consists only of the)END statement. SDF2/DM ignores any data that appears on lines following the)END statement.

Chapter 3. Panel processing statements

The)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC sections can contain the following types of statements:

Assignment
ELSE
EXIT
GOTO
IF
REFRESH —)REINIT and)PROC sections only
VER (verify) —)PROC sections only
VGET
VPUT

The assignment statement can use the following built-in functions:

LVLINE (last visible line)
TRANS (translate)
TRUNC (truncate)

The following types of data references can appear within panel section statements:

Dialog variable	A name preceded by an ampersand (&).
Control variable	A name preceded by a period (.). Control variables are described in Chapter 4, "Control variables" on page 100.
Literal value	A character string not beginning with an ampersand or period. A literal value can be enclosed in single quotes (' '). It must be enclosed in single quotes if it begins with a single ampersand or a period, or if it contains any of the following special characters: blank < (+ |) ; ¬ − , > : = ' "

A literal can contain substitutable variables, consisting of a dialog variable name preceded by an ampersand (&). The name and ampersand are replaced with the value of the variable prior to processing the statement. Trailing blanks are removed from the variable before the replacement. You can use a double ampersand to specify a literal character string starting with, or containing, an ampersand.

In the description of statements and built-in functions that follows, a "variable" can be either a dialog variable or a control variable. A "value" can be either type of variable or a literal value.

The Assignment statement

Syntax

variable **=** *value*

Examples:

```
.ALARM = YES
&DSN   = '''SYS1.MACLIB'''
&BB    = &C
```

The first example sets the control variable .ALARM to YES. The second example sets the variable DSN to the character string 'SYS1.MACLIB'. The third example sets variable BB to the value of variable C.

Each assignment statement must follow the rules for special characters given in Appendix, “Special characters” on page 229.

Built-in functions

The right-hand side of an assignment statement can contain the following built-in functions:

LVLINE This function has the following syntax:

Syntax

LVLINE(*dynamic-area-name* |
scrollable-area-name |
*scrollable-list-name***)**

It returns the number of lines in this area or list (the line number of the last visible line). For a scrollable area or list, this number includes the scrolling indicator line.

TRANS This function has the following syntax:

Syntax

TRANS(*variable before***,***after*... [**MSG=***msg*]**)**

It returns a translation of the current value of *variable*. If the value of the variable is *before*, it is translated to *after*. The character string specified for *after* may contain a variable name. In this case, the value is substituted when the statement is executed.

For example:

```
&A = TRANS(&XYZ Y,YES N,NO)
```

If the value of XYZ if Y, the value of A is YES. If the value of XYZ if N, the value of A is NO. If XYZ has any other value, the value of A is a blank. The translation does not affect the value of XYZ itself.

To cover the possibility that XYZ has a value other than Y and N, you can specify an asterisk. For example:

```
&A = TRANS(&XYZ Y,YES N,NO *,'?')
&A = TRANS(&XYZ Y,YES N,NO *,*)
```

The first example gives A a value of ? if XYZ is neither Y nor N. The second example gives A the value that XYZ has if XYZ is neither Y nor N.

Another way to deal with this situation is to issue a message when the value of XYZ is neither Y nor N. This is particularly useful if you want to prevent the user from entering an incorrect value. For example:

```
&A = TRANS(&XYZ Y,YES N,NO MSG=EMPX21)
```

If the value of XYZ is neither Y nor N, message EMPX21 is issued.

TRUNC This function can have either of the following syntaxes:

Syntax

TRUNC(*variable***,***number***)**
TRUNC(*variable***,***char***)**

TRUNC(*variable*,*number*) returns the specified number of characters of the variable value, starting at the left. All other characters are stored in the .TRAIL control variable. For example, the TRUNC(&XYZ,5) function returns the first 5 characters of the current value of XYZ and stores all remaining characters in .TRAIL.

TRUNC(*variable*,*char*) returns all the characters before the specified character. All characters after the specified character are stored in the .TRAIL control variable. For example, if XYZ has the value 3.25, the TRUNC(&XYZ,'.') function returns the value 3 and stores the value 25 in .TRAIL. The specified character (.) is not returned and is not stored in .TRAIL.

Before the TRUNC function is executed, the variable .TRAIL is reset to blanks. This means that you cannot perform the TRUNC function on the .TRAIL variable directly. For example:

```
&ERROR = TRUNC(.TRAIL,1)
```

would always return a blank, but:

```
&TEMP  = .TRAIL
&ERROR = TRUNC(&TEMP,1)
```

would return the first character of the value that was previously stored in .TRAIL.

For both the TRANS function and the TRUNC function, you can specify the same variable name on the right and left sides of the assignment statement. For example:

```
&XYZ = TRANS(&XYZ Y,YES N,NO)
```

You can also nest the TRANS and TRUNC functions. For example:

```
&XYZ = TRANS(TRUNC(&ZCMD,'.') 1,FILE 2,OPEN 3,QUIT)
```

This first truncates the value of ZCMD to remove everything after the period (if the ZCMD value contains a period). It then translates 1 to FILE, 2 to OPEN, and 3 to QUIT, and stores the value in XYZ. The value of ZCMD itself remains unchanged.

The ELSE statement

The ELSE statement begins a block of statements that are processed when the corresponding IF statement is not true. For more information, see "The IF statement" on page 93.

The EXIT statement

The EXIT statement ends processing of the panel definition section. All remaining statements in the section are ignored. Further processing of the panel continues normally.

The EXIT statement has no parameters. It normally appears within an IF or ELSE block, as described in "The IF statement" on page 93.

For example:

```
IF (&CUSTNAME=' ')
   .MSG = DGIE016
   EXIT
```

The GOTO statement

The GOTO statement transfers processing to a point later in the same panel definition section. All intervening statements in the section are ignored.

Syntax

GOTO *label*
  ⋮
label:
  ⋮

label Has the following characteristics:

- 1–8 characters in length
- First, or only, character must be A–Z or a–z
- Remaining characters, if any, must be A–Z, a–z, or 0–9

Processing passes to the line after the label line. The label line must appear after the GOTO line, in the same panel definition section. On the label line, the label is followed by a colon.

The label is translated to uppercase before being processed. Duplicate labels are ignored.

For example:

```
IF (&CUSTNAME=' ')
   .MSG = DGIE016
   GOTO ERROUT
  ⋮
ERROUT:
  ⋮
```

The label line does not need to be at the same level of indentation as the GOTO line. Ordinarily, the GOTO line will be indented within an IF or ELSE block and the label line will not be indented. If the label line appears within an IF block (which is not recommended), processing continues normally and any subsequent ELSE block at the same level of indentation is not processed.

The IF statement

An IF statement causes conditional processing.

Syntax

IF (*expression***)**
  ⋮

It can optionally be followed by an ELSE statement:

Syntax

IF (*expression***)**
  ⋮
[**ELSE**]
  ⋮

All statements following an IF statement and indented from the IF statement are executed if *expression* is true. If *expression* is not true, the statements following the IF statement are not executed but any statements following the ELSE statement are executed.

You can imbed an IF statement within an IF or ELSE block. IF statements can be nested a maximum of 15 levels deep.

Unlike in some programming languages, there is no END statement to end an IF or ELSE block. Instead, the level of indentation determines which statements are conditional:

- All statements following an IF statement that are indented further than the IF statement are part of the IF block.
- An ELSE statement must have the same level of indentation as the corresponding IF statement.
- All statements following an ELSE statement that are indented further than the ELSE statement are part of the ELSE block.
- The first statement after an IF or ELSE statement that has the same amount of indentation or less indentation ends the IF or ELSE block.

For example:

```
IF (&TEMP <= 0)
    &STATE = SOLID
    &NAME = ICE
ELSE
    IF (&TEMP >= 100)
        &STATE = GAS
        &NAME = STEAM
    ELSE
        &STATE = LIQUID
        &NAME = WATER
```

In this example, the variable TEMP represents a temperature in degrees Celsius. The variables STATE and NAME describe the behavior of water at that temperature.

The *expression* in an IF statement can be any of the following:

- A comparison of a variable against a value
- A comparison of a variable against a list of values
- A VER statement without the MSG keyword
- A combination of simpler expressions joined by AND or OR

Comparison against one value

The simplest form of expression for an IF statement is a comparison of a variable against a single value. The value itself can be the name of a variable.

These comparison operators can be used:

Equal	= or EQ
Not equal	¬= or NE
Greater than	> or GT
Less than	< or LT
Greater than or equal (not less)	>= or GE or ¬< or NL
Less than or equal (not greater)	<= or LE or ¬> or NG

If you use the literal version of an operator (such as EQ), specify at least one blank on each side of the operator. If you use the symbol version of an operator (such as =), blanks are optional. If you use the two-symbol version of an operator (such as ¬=), do not include a blank between the two symbols.

When comparing values, SDF2/DM uses a numeric comparison if both values are integers between −2147483648 and +2147483647. Thus, if the variable A has the value +1 and the variable B has the value 1, the expression IF (&A=&B) is true.

If either or both of the variables is not an integer between −2147483648 and +2147483647, SDF2/DM uses a character comparison in the EBCDIC collating sequence.

For both numeric comparison and character comparison, trailing blanks are ignored.

Comparison against a list of values

With the equal and not-equal operators, you can specify a list of values. For example:

```
IF (&ZCMD = 1,2,3,4,5)
IF (&ZCMD NE 1,2,3,4,5)
```

The first example is true if ZCMD has any of the listed values. The second example is true if ZCMD has none of the listed values.

You cannot specify a list of values for the other operators (greater than, less than, greater than or equal to, or less than or equal to).

VER statement on an IF statement

The expression on an IF statement can be a VER statement. This VER statement has the same syntax as a standalone VER statement, except that you cannot use the MSG keyword. For the syntax of a VER statement, see “The VER statement” on page 97.

Examples:

```
IF(VER (&ZCMD NB,LIST,1,2,3,4,5))
IF(VER (&ZCMD NUM))
```

The first example is true if ZCMD is not blank and has any of the listed values (the numbers from 1 through 5). The second example is true if ZCMD is blank or consists entirely of the digits 0 through 9.

The VER statement can be split over more than one line, but the VER statement and the left parenthesis that follows it must be on the same line.

Expressions joined by AND or OR

The expression on an IF statement can consist of 2 to 255 simpler expressions joined by AND or OR. AND and OR are sometimes called Boolean operators or logical operators. These logical operators can be used:

& *or* AND	True if both of the expressions before and after it are true
| *or* OR	True if the either or both of the expressions before and after it are true

Each logical operator must be preceded and followed by at least one blank.

The expressions are evaluated from left to right. An AND operator takes precedence over an OR operator. For example:

```
IF (exp1 OR exp2 AND exp3)
```

is true if *exp1* is true, and is also true if both *exp2* and *exp3* are true.

You cannot give OR precedence over AND. To achieve the same result, you can break the statement into two IF statements:

```
IF (exp1 OR exp2)
    IF (exp3)

⋮
```

or you can reorganize the statement as follows:

```
IF (exp1 AND exp3 OR exp2 AND exp3)
```

The REFRESH statement

A REFRESH statement causes the value of one or more variables to be updated on the panel.

Syntax

REFRESH *varlist*

To update one variable, give the variable name in *varlist*. To update more than one variable, list the variable names within parentheses. To update all input and output variables, specify an asterisk.

For example:

```
REFRESH (SEL, RNAME)
REFRESH(*)
```

The first example updates the value of SEL and RNAME on the panel. The second example updates the value of every input or output field on the panel.

When a panel is initially displayed, each variable name is replaced by its value (stripped of trailing blanks). By specifying a REFRESH statement in the)REINIT or)PROC section, you can update the variable value when the panel is redisplayed. Otherwise, the panel continues to display the previous value of the variable.

The REFRESH statement cannot appear in the)INIT section, where it would be meaningless. (The initial display of the panel reflects the value of each variable after the)INIT section is processed.)

The VER statement

A VER statement verifies that the value of a variable meets the specified criteria.

Syntax

VER (*variable* **,NB** [**,MSG=***msg*]**)**
VER (*variable* [**,NB**]**, ALPHA** [**,MSG=***msg*]**)**
VER (*variable* [**,NB**]**, NUM** [**,MSG=***msg*]**)**
VER (*variable* [**,NB**]**, HEX** [**,MSG=***msg*]**)**
VER (*variable* [**,NB**]**, RANGE,** *lower*, *upper* [**,MSG=***msg*]**)**
VER (*variable* [**,NB**]**, LIST,** *value1*, *value2* ... [**,MSG=***msg*]**)**

variable	Is the name of the panel field whose value you are verifying.
NB	(or the full word NONBLANK) specifies that the variable cannot be blank. This can be combined with other criteria. (If you do not specify NB, a blank value will pass the ALPHA, NUM, HEX, RANGE, and LIST criteria.)
ALPHA	The value must contain only A–Z, a–z, #, $, or @. It cannot contain blanks, unless the field is completely blank.
NUM	The value must contain only 0–9. It can contain leading blanks.
HEX	The value must contain only 0–9, A–F, or a–f. It can contain leading blanks.
RANGE	The value must a number in the range starting with *lower*, up to and including *upper*. The value can contain a leading + for a positive number or − for a negative number. The *lower* and *upper* values can contain up to 10 digits each, not counting the + or − sign. Any extra characters cause a conversion error.
LIST	The value must be one of the values in the list.
msg	The message ID of the message that will be displayed if the variable does not meet the verification criteria. If no message ID is specified, SDF2/DM displays a message based on the type of verification.

The VER statement is typically used in the)PROC section to verify the data stored in a dialog variable. Verification of an input variable is performed after the value is stored in the function pool or shared pool. The current rules for padding, justification, and VDEFINE apply to the value that is stored.

Even if a VER fails, the panel's)PROC and)REINIT statements are processed.

Example

Figure 47 shows a panel definition that includes VER statements in the)PROC section.

```
)BODY
%----------------------------  EMPLOYEE RECORDS  ------------------------------
%COMMAND===>_ZCMD                                                              %
+
%EMPLOYEE SERIAL: &EMPSER
+
+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+
+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                              +
+     LINE 2 %===>_ADDR2                              +
+     LINE 3 %===>_ADDR3                              +
+     LINE 4 %===>_ADDR4                              +
+
+   HOME PHONE:
+     AREA CODE   %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +
+
)INIT
  IF (&PHA = ' ')
    &PHA = 301
  &TYPECHG = TRANS (&TYPECHG N,NEW U,UPDATE D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  VER (&TYPECHG,LIST,N,U,D,MSG=EMPX210)
  VER (&LNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NUM)

)END
```

Figure 47. Panel definition example with verification

The VGET statement

A VGET statement gets the value of one or more variables from the shared pool or from the profile pool. A VGET statement in a panel definition is equivalent to the VGET dialog management service in an application prototype. For more information, see “VGET — Retrieve variables from a pool or profile” on page 211.

Syntax

VGET *varlist* [**<u>ASIS</u>**|**SHARED**|**PROFILE**]

To get one variable, give the variable name in *varlist*. To get more than one variable, list the variable names within parentheses.

For example:

```
VGET (SEL, RNAME) PROFILE
```

The values of SEL and RNAME are copied from the profile pool.

The VPUT statement

A VPUT statement copies the value of one or more variables to the shared pool or to the profile pool. A VPUT statement in a panel definition is equivalent to the VPUT dialog management service in an application prototype. For more information, see “VPUT — Update variables in the shared, profile, or global pool” on page 213.

Syntax

VPUT *varlist* [**<u>ASIS</u>**|**SHARED**|**PROFILE**]

To store one variable, give the variable name in *varlist*. To store more than one variable, list the variable names within parentheses.

For example:

```
VPUT (SEL, RNAME) PROFILE
```

The values of SEL and RNAME are stored in the profile pool.

Chapter 4. Control variables

You can use control variables to change the state of the panel or to determine its current state. Control variables can be used only in the)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC sections of a panel definition. They can be used in these ways:

- On the left side of an assignment statement, to change the state of the panel. For example:

  ```
  .ALARM = YES
  ```

- On the right side of an assignment statement, to save the current state in a variable. For example:

  ```
  &VAR = .ALARM
  ```

- In an IF statement, to take conditional action depending on the current state of the panel. For example:

  ```
  IF (.ALARM = YES)
  ```

The following control variables are available:

.ALARM

Can have the value YES, NO, null, or blank. YES causes the terminal alarm to sound when the panel is displayed. A value of NO, null, or blank causes the terminal alarm to be silent when the panel is displayed.

This control variable can also be set to a variable whose value is YES, NO, null, or blank.

Examples:

```
.ALARM = YES
.ALARM = &VAR
```

In the first example, the alarm sounds when the panel is displayed. In the second example, the alarm sounds if the value of VAR is YES.

If the panel is displayed with a message that has .ALARM = YES, the alarm sounds regardless of the setting of .ALARM in the panel assignment statement.

.ATTR and .ATTRCHAR

Override the attributes of a specific field or override the attributes of every field that is identified with a specific attribute character.

Syntax

.ATTR (*field***) = '***keyword***(***value***)** [*keyword***(***value***)** ...]**'**
.ATTRCHAR (*char***) = '***keyword***(***value***)** [*keyword***(***value***)** ...]**'**

For .ATTR, *field* can be any of the following:

- The name of an input or output field that occurs in a)BODY or)AREA section.
- The name of an action-bar choice. Only the type can be changed from AB to ABUC or vice versa.

- The name of a pull-down choice, if the statement appears within an)ABCINIT or)ABCPROC section. Only the type can be changed from PD to PDUC or vice versa.
- .CURSOR, which means the field in which the cursor is currently positioned.
- The name of a variable preceded by an ampersand, where the variable contains either .CURSOR or the name of a field or choice.

For .ATTRCHAR, *char* can be any of the following:

- An attribute character that appears in the)ATTR section: this is either the character itself or the 2-digit hexadecimal number that corresponds to the character. This character must be enclosed in single quotes if it is one of the following special characters:

  blank < (+ |) ; ¬ – , > : + ' "

- The name of a variable preceded by an ampersand, where the variable contains an attribute character or its hexadecimal representation.

Each *keyword* and *value* pair must be valid for the field type. Only the specified keyword values are overridden. Any other keyword values remain unchanged.

Examples:

```
.ATTR (.CURSOR) = 'COLOR(YELLOW) HILITE(REVERSE)'
.ATTR (&FLD)    = 'HILITE(&HLTE)'
.ATTRCHAR ('¬') = 'COLOR(BLUE)'
```

In the first example, the color and highlighting of the field containing the cursor is overridden. In the second example, the variable FLD contains the name of a field, and the variable HLTE contains the new highlighting value for the field. In the third example, all fields that are identified by the attribute character ¬ are changed to blue.

Overriding the cursor field (.CURSOR) and the alternative long or short message field attributes can be particularly useful if the panel is being redisplayed because of a translation or verification failure. If such a failure occurs, the cursor is automatically placed on the field in error, and the message ID to be displayed is automatically placed in the message area.

For example, if SMFIELD is specified on the)BODY statement as the alternative short message field, a)REINIT section could be specified as follows:

```
)REINIT
 .ATTR (.CURSOR) = 'COLOR(RED) HILITE(USCORE)'
 .ATTR (SMFIELD) = 'HILITE(BLINK)'
```

This would cause the field in error to be redisplayed in red and underscored and the error message to blink.

When a field attribute is overridden in the)INIT section of a panel, the override remains in effect if the panel is redisplayed, unless the attribute is again overridden by another statement in the)REINIT section. An attribute override in the)PROC or)REINIT section of the panel remains in effect for only a single redisplay of that panel, should a redisplay occur. Thus, one field at a time can be highlighted as errors are found. When the user corrects the error, the field reverts to its normal attributes.

The following restrictions apply to the .ATTR and .ATTRCHAR control variables:

- You cannot override the COLOR, INTENS, or HILITE keyword for a CUA attribute type (see "CUA attribute types" on page 50). Users can change the values of these keywords interactively, by means of the CUAATTR command.
- You can change the CUA attribute type of fields by specifying the CUATYPE(*type*) keyword.
- You can override the CAPS, JUST, PAD, PADC, SKIP, or NUMERIC keyword for a CUA attribute type only if the keyword applies to this CUA attribute type. For example, none of these keywords applies to the AB attribute type.
- In general, you cannot change the value of the TYPE keyword from one CUA attribute type to another. For example, if you attempt to change the TYPE of a field from LI to SAC or from NEF to INPUT, a dialog error will result. Exceptions are:
  - TYPE keyword values that have a field type of INPUT can be overridden with TYPE(EE) — error emphasis
  - TYPE keywords AB and ABUC can override each other
  - TYPE keywords PD and PDUC can override each other
  - TYPE keywords SAC and SUC can override each other
- You can change the TYPE keyword from INPUT to OUTPUT or to any CUA attribute whose field type is either input or output.
- You can change the TYPE keyword from OUTPUT to INPUT or to any CUA attribute whose field type is either input or output.

For example, you can change TYPE(INPUT) to TYPE(NEF). If you try to change TYPE(INPUT) or TYPE(OUTPUT) to TYPE(AB) or TYPE(RP), an error condition results.

.CURSOR and .CSRPOS and .CSRIDX

The value of .CURSOR is the name of the field or dynamic area that contains the cursor. The .CURSOR control variable overrides any cursor position specified on the DISPLAY service request.

The .CSRPOS control variable is valid only if the cursor is on an input/output or dynamic area field. The value of .CSRPOS is a number representing the position of the cursor within this field, starting with 1 for the beginning of the field.

The .CSRIDX control variable is valid only when the cursor is on an input/output field of a scrollable list. The value of .CSRIDX is a number representing the index for the array variable referred to by .CURSOR. For a scrollable list .CSRIDX denotes the row number.

Examples:

```
.CURSOR = XYZ
```

puts the cursor at the beginning of the field or dynamic area named XYZ.

```
.CURSOR = XYZ
.CSRPOS = 3
```

puts the cursor two spaces to the right of the beginning of the field or dynamic area named XYZ.

```
&VAR1 = .CURSOR
&VAR2 = .CSRPOS
```

stores the cursor position in variables VAR1 and VAR2. VAR1 contains the field or dynamic area name; VAR2 contains the cursor position within the field.

If the control variable .CURSOR is not explicitly initialized, or if it is set to blank, the initial field in which the cursor is positioned is determined as follows:

1. The panel body is scanned from top to bottom, and the cursor is placed at the beginning of the first input field that meets the following criteria:
   - It must be the first or only input field on a line.
   - It must not have an initial value; that is, the corresponding dialog variable must be null or blank.
   - It must not have a field name of ZCMD.
   - It must not be the action-bar selection field.
2. If these criteria are not met in the panel body, the scrollable areas are searched using the same criteria.
3. If the criteria are still not met, the cursor is placed on the first input field in the panel body or scrollable area.
4. If the panel has no input fields, the cursor is placed at the top left-hand corner of the screen.

The cursor is automatically placed at the beginning of the field that was last referred to in any panel definition statement when a message is displayed because of:

- A verification failure that sets .MSG
- A .MSG=*value* condition in a TRANS
- An explicit setting of .MSG

.FHELP
This has only the format of a control variable, but is in fact a statement for defining a field help panel associated with a specific field. For detailed information see “The)HELP section” on page 87.

.HELP
The value of this control variable is the name of the extended help panel. Extended help (also called panel help) is displayed when a user requests extended help. It is also displayed when a user requests help and no other help is available.

Example:

```
.HELP = DGITE
```

.MSG

The value of this control variable is the name of a message that is displayed. This control variable is typically used in the)PROC section when an error condition is detected.

Example:

```
.MSG = DGIE016
```

In this example, message DGIE016 will be displayed.

The .MSG variable can be set explicitly in an assignment statement, or automatically by a VER statement or the MSG keyword in a TRANS function call.

.RESP

Can have the value ENTER or END.

If the user enters an END or RETURN command, this control variable is set to END. Otherwise, it is set to ENTER.

You can use this control variable to determine which key the user pressed. For example:

```
IF (.RESP = END)
```

You can set the value of this control variable in the)PROC section. This is particularly useful if verification fails (or .MSG is set) and you want to redisplay the panel with a message even if the user entered END or RETURN.

For example:

```
VER(&VAR1 NB)
IF(.MSG¬='')
   .RESP=ENTER
```

Setting .RESP in the)INIT or)REINIT section of the panel definition has no effect if a message is being displayed.

The)INIT or)REINIT section can be coded with the following statements to ensure that the panel is not displayed, regardless of whether a message was specified on the DISPLAY service request:

```
  IF (.MSG ¬= &Z)
    .MSG = &Z
  .RESP = END
```

In the previous example, the)INIT and)PROC sections would be executed, but the panel would not be displayed. The value of .MSG would be reset to a null value.

.TRAIL

Contains the remainder following a truncate (TRUNC) operation. If the contents of a variable are truncated to a specified length, all remaining characters are stored in .TRAIL. If the contents of a variable are truncated at the first occurrence of a special character, the remaining characters following the special character (not including the special character itself) are stored in .TRAIL.

.ZRP

Can be set to a list of variable names in the)BODY,)AREA, or)LIST section. It must start in column 1. These variable names replace every occurrence of a reference phrase in the section in which it is specified. The purpose is to assign meaningful names to reference phrases. .ZRP must not be set to a dialog variable. If you specify more than one name, the value of ZRP is a list of variable names, separated by blanks or commas, enclosed in quotes and parentheses. For example:

```
.ZRP = '(NAME TOWN ZIPCODE)'
```

.ZSF

Can be set to a list of variable names in the)BODY,)AREA, or)LIST section. It must start in column 1. These variable names replace every occurrence of a selection field in the section in which it is specified. The purpose is to assign meaningful names to selection fields. .ZSF must not be set to a dialog variable. If you specify more than one name, the value of .ZSF is a list of variable names, separated by blanks or commas, enclosed in quotes and parentheses. For example:

```
.ZSF = '(NAME TOWN ZIPCODE)'
```

.ZVARS

Can be set to a list of variable names in the)INIT section. These variable names replace every occurrence of the placeholder variable Z in the)BODY and)AREA sections of the panel definition.

.ZVARS can also be specified in a)BODY,)AREA, or)LIST section. In this case, it must start in column 1 and must not contain variable names. This might be needed if, for example, a)BODY section included text, with an .INCLUDE statement, containing placeholder variables Z.

If you specify more than one Z variable, the value of ZVARS is a list of variable names, separated by blanks or commas, enclosed in quotes and parentheses.

Variable names are substituted in the order in which the Z variables appear in the panel definition, which is not necessarily the order in which they will appear on the panel. All variables in the)BODY section are substituted, then the variables in the first)AREA section, then the variables in the next)AREA section, and so on.

Example:

```
)BODY
---------------------------  TITLE LINE  -----------------------------------
%COMMAND===>_ZCMD                                                          %
%  .
   .
   .
   .
+   TYPE  %===>_Z+    (A for alpha, N for numeric)
+   LENGTH%===>_Z +   (0 to 99)
+   OFFSET%===>_Z +   (0 to 99)
   .
   .
   .
)INIT
  .ZVARS = '(TYPFLD LNGFLD OFFSET)'
```

In this example of Z-variable placeholders, the first input field is 1 character long and the other two input fields are each 2 characters long. The names of these three fields are TYPFLD, LNGFLD, and OFFSET.

.ZVARS can also be set to a dialog variable whose value is a valid name list. For example:

```
.ZVARS = &NLIST
```

where the value of NLIST is (TYPFLD LNGFLD OFFSET)

Control variables are automatically reset to blank when the panel display service first receives control. If .MSG, .CURSOR, and .CSRPOS are still blank after processing of the initialization section, they are set to the values, if any, passed by the calling sequence for an initial message or cursor placement. Under certain conditions, processing of the initialization section is bypassed.

When .CURSOR, .CSRPOS, .MSG, and .RESP have been set to nonblank by panel processing, they retain their initial values until the panel is displayed, or redisplayed, at which time they are reset.

The control variables

.CURSOR
.HELP
.MSG
.RESP

have a length of 8 bytes. When set in an assignment statement to a longer value, the value is truncated on the right. If these control variables are tested in a conditional expression, the compare value (literal or dialog variable) must not be longer than 8 bytes.

Chapter 5. Keylists

This chapter explains how to establish keylists for your application prototype panels. A *keylist* specifies what happens when a user presses a function key while a panel is being displayed.

There are two ways to specify a keylist:

- Write a)KEYLIST section in the panel definition. This is called an *imbedded keylist*. It starts with a)KEYLIST statement and ends with the beginning of the next panel section.
- Put the)KEYLIST section into a file of its own. This is called a *standalone keylist*. It starts with a)KEYLIST statement and ends with an)END statement. To use a standalone keylist in a panel, specify the keylist name on the)PANEL statement.

Each panel can have only one keylist. If the panel definition has a)KEYLIST section, this keylist is used. Otherwise, the keylist specified on the)PANEL statement is used. If there is no)KEYLIST section and no keylist is specified on the)PANEL statement, the default keylist is used.

Standalone keylists and imbedded keylists have the same syntax. The keylist begins with the)KEYLIST statement:

Syntax

)KEYLIST [**HELP(***panelname***)**]
[**LABELS(<u>YES</u>|NO)**]

Each function key has an entry in the following syntax:

Syntax

KEY(*keyid***) CMD(***command***) FKA(<u>YES</u>|NO|***varname***) TEXT(***keytext***)**

)KEYLIST
: Starts the keylist definition section. This section can be imbedded in a panel definition (imbedded keylist) or can be the only section in a file or member (standalone keylist).

HELP(*panelname***)**
: Specifies which help panel is to be displayed when the KEYSHELP command is entered. The help panel can also be defined by setting the ZKEYHELP variable.

LABELS(<u>YES</u>|NO)
: Specifies whether labels (such as F3=) are to be generated by SDF2/DM. If you specify NO, only the text specified with the keyword TEXT(...) is displayed. The default is YES.

KEY(*keyid*)
Specifies the function key. For each function key used on a panel, a KEY entry must be defined. The *keyid* value can be specified as a number, or as a number preceded with F or PF. For example, KEY(3), KEY(F3), and KEY(PF3) are identical.

If an undefined or inactive function key is pressed, SDF2/DM displays an error message.

CMD(*command*)
Specifies which command is to be performed when the appropriate key is pressed. The *command* variable can contain from 1 to 8 characters. You can specify a variable name preceded by an ampersand (&).

FKA(<u>YES</u>|NO|*varname*)
Specifies whether the key text is to be shown in the function key area and whether the key is to be active. You can specify a variable name preceded by an ampersand (&). The variable must contain Y[ES], N[O], or S[KIP]. You can then dynamically hide or deactivate a function key.

YES	The function key is active and shown in the function key area.
NO	The function key is active but not shown in the function key area.
SKIP	The function key is inactive and not shown in the function key area.

TEXT(*keytext*)
The text to be displayed in the function key area. You can specify a variable name preceded by an ampersand (&).

How keylists are displayed

Keylists are displayed as follows:

- The maximum number of function keys that can be formatted on each line is displayed.
- Each displayed function key definition appears as **F***nn***=***label* or **F***n***=***label* (where *nn* or *n* is the numeric value of the function key).

For example:

```
COMMAND ===>
F1=HELP     F2=TOP      F3=END      F4=RETURN   F5=BOTTOM F6=LOCATE
F7=UP       F8=DOWN     F9=PRINT   F10=LEFT    F11=RIGHT F12=RETRIEVE
```

How keylists are processed

For each function key, you can specify a KEY statement. When a user presses the function key, the effect is the same as typing the character string that appears in the CMD keyword of the KEY statement and pressing Enter. The application prototype cannot determine whether a command was entered by a function key or by typing in the command field.

For example, if you specify:

```
KEY(F7)  CMD(UP)       FKA(YES)       TEXT('BKWD ')
```

a user can either press the F7 key or type UP and press the Enter key.

If the user types something in the command field and then presses a function key, the text in the CMD keyword, followed by a blank, followed by the text in the command field is processed. For example, if a user types 15 and presses F7, the command UP 15 is processed.

If a command is not recognized by SDF2/DM, it is passed on to the application prototype.

If there is no keylist, you can use the KEYS command to set the values of function keys 1 through 12. Any function key from 13 through 24 always has the same effect as the corresponding function key from 1 through 12.

If there is a keylist, the KEYS command sets the values of variables ZPF01 through ZPF12. (This has an effect only if you use these variables in your keylist.) For example, in Figure 48 on page 110 these variables control function keys 13 through 24. F1 is permanently set to HELP, but the user can change the value of F13 by setting the value of ZPF01.

Figure 49 on page 111 shows, as part of a complete panel definition, an example of an imbedded keylist.

Examples of keylists

Figure 48 shows a stand-alone keylist. Figure 49 on page 111 shows an embedded keylist.

```
)KEYLIST HELP(DGIRKBRS)            /* for panels with Browse key    */
 key(f1)  cmd(help)     fka(yes)      text('Help ')
 key(f2)  cmd(browse)   fka(yes)      text('Browse ')
 key(f3)  cmd(exit)     fka(yes)      text('Exit ')
 key(f4)  cmd(process)  fka(yes)      text('Process ')
 key(f5)  cmd(rgtleft)  fka(yes)      text('Rgtleft ')
 key(f6)  cmd(rfind)    fka(yes)      text('Rfind ')
 key(f7)  cmd(up)       fka(yes)      text('Bkwd ')
 key(f8)  cmd(down)     fka(yes)      text('Fwd ')
 key(f9)  cmd(swap)     fka(&DGISWOPT) text('Swap ')
 key(f10) cmd(actions)  fka(yes)      text('Actions ')
 key(f11) cmd(cretriev) fka(yes)      text('CRetrieve ')
 key(f12) cmd(cancel)   fka(yes)      text('Cancel')
 key(f13) cmd(&zpf01)   fka(no)       text('')
 key(f14) cmd(&zpf02)   fka(no)       text('')
 key(f15) cmd(&zpf03)   fka(no)       text('')
 key(f16) cmd(&zpf04)   fka(no)       text('')
 key(f17) cmd(&zpf05)   fka(no)       text('')
 key(f18) cmd(&zpf06)   fka(no)       text('')
 key(f19) cmd(&zpf07)   fka(no)       text('')
 key(f20) cmd(&zpf08)   fka(no)       text('')
 key(f21) cmd(&zpf09)   fka(no)       text('')
 key(f22) cmd(&zpf10)   fka(no)       text('')
 key(f23) cmd(&zpf11)   fka(no)       text('')
 key(f24) cmd(&zpf12)   fka(no)       text('')
)END
```

Figure 48. Example of a standalone keylist

```
)PANEL
)ATTR
  { TYPE(PT)                        /* panel title   (left brace) C0 */
  ¢ TYPE(OUTPUT) CUATYPE(SI) JUST(ASIS)  /* scroll indicators     4A */
  # TYPE(SI)                        /* scrollable area            7B */
  _ TYPE(NEF) CAPS(ON)              /* normal entry f.w. caps on  6D */
  ¬ TYPE(NEF)                       /* normal entry field         5F */
  < TYPE(TEXT) COLOR(WHITE) INTENS(HIGH) /* Command ===>             */
  $ TYPE(FP)                        /* field prompt               5B */
  @ TYPE(MSGS)                      /*                               */
  ■ TYPE(MSGI)                      /*                               */
  | AREA(DYNAMIC) EXTEND(OFF) SCROLL(ON) /*                       4F */
 21 TYPE(DATAIN)  CUATYPE(NEF)      /*                            21 */
 22 TYPE(DATAOUT) CUATYPE(VOI)      /*                            22 */
)BODY WINDOW(76,18)
                            {DGI Messages+        @DGISMSGS           +
<Command ===>¬ZCMD                                              $Scroll_Z
+                                                                #&DGIS¢Z +
|DGIDMSGS                                                             |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
“DGILMSG                                                            %
)INIT
 .HELP = DGIRMSGS
 .ALARM = &DGIALARM
 &ZWINTTL=''
 .ZVARS = (DGISCR DGISCRLI)
 &DGISCRLS='SKIP'                   /* assume no scrolling possible  */
 &IPVS=''
 IF(&DGISCRLI¬='')                  /* if scroll up/down possible    */
   &DGISCRLS='YES'                  /* show scroll keys              */
   &SDF2='More:'                    /* show scroll indicator         */
)PROC
 &DGIDEPTH=LVLINE(DGIDMSGS)
)HELP
 FIELD(DGILMSG ) PANEL(&DGIMH)
)KEYLIST
 key(f1)  cmd(help)     fka(yes)       text('Help ')
 key(f3)  cmd(end)      fka(yes)       text('Exit ')
 key(f7)  cmd(up)       fka(&DGISCRLS) text('Bkwd ')
 key(f8)  cmd(down)     fka(&DGISCRLS) text('Fwd ')
 key(f12) cmd(end)      fka(yes)       text('Cancel')
)END
```

Figure 49. Example of an imbedded keylist

Chapter 6. Messages

This chapter describes how to create messages. A message can be displayed in any of the following situations:

- You use the SETMSG service.
- You use the DISPLAY service with the MSG keyword.
- In an assignment statement, you give a value to .MSG.
- In a VER statement with the MSG keyword, verification fails.
- In a TRANS function call with the MSG keyword, translation fails.

Only one message at a time can be displayed on a panel. When a panel is displayed, the following sequence determines which message is displayed:

1. The message specified on the first SETMSG statement has the lowest priority.
2. Any message specified on a subsequent SETMSG statement overrides the previous SETMSG, unless you specify the COND keyword.
3. Any message specified on the DISPLAY statement overrides any SETMSG statement.
4. Any message specified in the)INIT section of the panel definition overrides any DISPLAY statement.

Information about the current message is stored in the system variables listed in Figure 57 on page 226. You can also get information about a message, without displaying the message, by using the GETMSG service.

Message syntax

Messages are stored in message members. A message member must start with the)MSG header statement and must end with the)END statement. The name of a message member contains up to seven characters. It consists of from 1 to 5 alphabetic characters (A–Z, #, $, or @) followed by two numeric characters (0–9).

Within a message member, each message has a unique message ID. A message ID consists of the message member name followed by one alphanumeric character (usually a number) and optionally followed by one alphabetic suffix character.

The maximum length of a message ID is 8 characters. If you use suffix characters in your message IDs, the maximum length of a message member name is 6 characters.

For example the message IDs DGI21*xx* are contained in the message member DGI21.

Each message definition consists of two or more lines.

```
Line 1:
msgid['short message'] [.HELP=panel] [.ALARM=YES|NO]
[.WINDOW=RESP|NORESP] [.TYPE=NOTIFY|WARNING|ACTION|CRITICAL]

Additional long message text lines—optional

Line 2:
['long message' [+] ]
Line 3:
['long message' [+] ]
Line n:
['long message' ]
```

Although the syntax of line 1 is split over two lines, the actual text in the message member must be contained on one line. If the long message is too long for one line, it can be split into several lines as described below.

You can optionally insert blank lines or comment lines between the end of one message and the beginning of the next message. A comment begins with the characters /* starting in column 1.

In the first line, the fields must be separated by at least one blank. One or more blanks can optionally occur on either side of an equal sign (=).

For an explanation of how to include special characters in the message text, see Appendix, "Special characters" on page 229.

You can use variables to represent the value of a keyword, and you can include variables within the message text. For example:

```
'VOLUME &VOL NOT MOUNTED'  .HELP = &H  .ALARM = &A
```

Each variable name is replaced by its value immediately before the message is displayed. After variable substitution, the short message is truncated to 24 characters and the long message is truncated to 512 characters.

msgid
: A unique identifier of up to 8 characters. It consists of the message member name (1–5 alphabetic characters followed by two numeric characters), followed by one alphanumeric character, optionally followed by one alphabetic suffix character. The message ID must begin in column 1.

short message
: A short version of the message. The maximum length of a short message is 24 characters. It must be enclosed in single quotes.

  If you specify a short message, the long message is not initially displayed. If the user requests help while a short message is being displayed, the long message is then displayed.

  The location of the short message is as follows:

  - The default short message area is the last 24 spaces on the first line of the panel or, if an action bar is defined, the last 24 spaces on the first line after the action-bar separator line.
  - You can override the default by specifying on the)BODY statement which field is the short message area.

- If you specify a value for the .WINDOW keyword or you specify .TYPE=CRITICAL, the short message is displayed in a pop-up window, not in the short message area.
- If an error occurs within a pull-down or a pop-up window, the short message area might not be long enough for the short message. In this case, the short message is displayed in a pop-up window.

.HELP=*panel*
The name of a help panel that is to be displayed when the user requests help for the long message. If you specify .HELP=*, or if you do not specify a value for .HELP, extended help for the panel is displayed.

.ALARM=YES|NO
YES means that an alarm is to be sounded when the message is displayed. NO means that an alarm is not to be sounded, unless .ALARM is set to YES in the panel definition. The default is NO.

If you specify .ALARM=YES and do not specify a .TYPE value, the message is displayed in yellow, high-intensity text. If you specify .ALARM=NO and do not specify a .TYPE value, the message is displayed in white, high-intensity text.

.WINDOW=RESP|NORESP
Causes the message to be displayed in a pop-up window. If you do not specify a value for .WINDOW, the short message (if any) is displayed in the short message area and the long message is displayed in the long message area.

.WINDOW=RESP means that the user must respond to the message (by pressing the Enter key or a function key) before working with the underlying panel. .WINDOW=NORESP means that the user can work with the underlying panel without responding to the message.

If the message is displayed because of the DISPLAY or SETMSG service, the MSGLOC keyword determines the location of the pop-up. If the message is displayed because of the VER statement or the TRANS function, the message pop-up is placed beside the field that is being validated, if possible.

If no message location can be determined by the previous rules, or if there is no room for the message at that location, the message pop-up is displayed at the bottom of the screen or below the active pop-up window, if one exists.

.TYPE=NOTIFY|WARNING|ACTION|CRITICAL
Identifies the type of the message. The .TYPE keyword overrides any value that you specify for .ALARM. If you specify .TYPE=CRITICAL, the message is displayed as if you specified .WINDOW=RESP, and the actual value of .WINDOW is ignored.

Figure 50 summarizes the characteristics of the different types of messages. These color and highlighting characteristics are the default values. You can change them by using the CUAATTR command.

Type	Color	Intensity	Placement	Response	Alarm
NOTIFY	White	High	Message area	Optional	Off
WARNING	Yellow	High	Message area	Optional	On
ACTION	Red	High	Message area	Optional	On
CRITICAL	Red	High	Pop-up window	Required	On

Figure 50. Message characteristics

long message

The long text of the message, enclosed in single quotes. It can contain up to 512 characters and must begin in column 1. You can split the long message into several lines by typing a closing quote, at least one blank space, and the + continuation character. For example:

```
DGIX001 'short message text'
'Long message text' +
' continued over ' +
'several lines.' +
'The maximum length is ' +
'512 bytes.'
```

If no short message exists, the long message is displayed. If a short message exists, it is displayed first; the long message is displayed only if the user requests help.

The location of the long message is as follows:

- The default long message area is the line after the command line if the command line is at the top of the panel, or the line before the command line if the command line is at the bottom of the panel.
- You can override the default by specifying which field is the long message area on the)BODY statement.
- If you specify a value for the .WINDOW keyword or if you specify .TYPE=CRITICAL, the long message is displayed in a pop-up window, not in the long message area.
- If the text is too long for the long message area, it is displayed in a pop-up window.

How message pop-ups are displayed

If you specify .TYPE=CRITICAL, or if you specify any value for .WINDOW, the message is displayed in a pop-up window.

If the message text is longer than the width of the pop-up window, the text is wrapped to the next line. The text is split only at blanks. If a single word is longer than the width of the pop-up window, the window is expanded to the length of the word. If the word is longer than the maximum window size (screen width minus 3), the word is split and continued on the next line.

When a message pop-up window is displayed, the user can cancel the pop-up by placing the cursor inside the pop-up and requesting CANCEL or ENTER. A message pop-up window is also canceled by the WINDOW command. This removes the pop-up without submitting the underlying panel for processing.

Example

Figure 51 shows an example of a message member. Because the first six characters of the messages in this example are EMPX21, the member name must also be EXMP21.

```
)MSG

EMPX210   'INVALID TYPE OF CHANGE' .HELP=PERS033   .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'

EMPX213    'ENTER FIRST NAME'       .HELP=PERS034  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX214    'ENTER LAST NAME'        .HELP=PERS034  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX215    'ENTER HOME ADDRESS'     .HELP=PERS035  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX216    'AREA CODE INVALID'      .ALARM=YES
'AREA CODE &PHA IS NOT DEFINED. PLEASE CHECK THE PHONE BOOK.'

EMPX217    '&EMPSER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE'

EMPX218    '&EMPSER UPDATED'
'RECORDS FOR &LNAME, &FNAME &I UPDATED'

EMPX219    '&EMPSER DELETED'
'RECORDS FOR &LNAME, &FNAME &I DELETED'

)END
```

Figure 51. Sample messages

Chapter 7. Defining file-tailoring skeletons

SDF2/DM skeleton definitions are stored in a library member and accessed through the SDF2/DM file-tailoring services. You create or change skeletons by editing the member directly. SDF2/DM interprets the skeletons during execution. Two types of records can appear in the skeleton file:

Data records	A continuous stream of intermixed text, variables, and control characters that are processed to create an output record.
Control statements	Control the file-tailoring process. Control statements start with a right parenthesis in column 1 and a nonblank character in column 2. (The right parenthesis is the default control statement control character.)

A)DEFAULT control statement can be used to define the set of control characters.

A file-tailoring skeleton must start with a)SKELETON control statement and end with an)END statement.

Considerations for data records

If variable substitution results in an output record larger than 80 characters, file tailoring terminates and a message is displayed.

Any blank data records in the input data are deleted from file-tailoring output. However, the)BLANK control statement can be used to produce blank lines in the output file.

Control characters

The following characters have special meanings:

) Control statement control character
The right parenthesis defines the following:

- The start of a control statement when placed in column 1 and followed by a nonblank character in column 2.
- The start of a data record when placed in column 1 and followed by a blank in column 2.

& Variable name control character
The ampersand indicates the start of a variable name. The value of the corresponding dialog variable is substituted in the output record. A value of all blanks is treated as null. Any of the following characters explicitly delimits the end of a variable name:

Blank ¢ < (+ | & ! *) ; ¬ - / , % _ > : ' = "

? Continuation record control character
The question mark indicates a continuation record when placed in the last input column of the record that is to be continued. The question mark is used as a continuation character when more than one input record maps to a single output record. If any character other than a question mark appears in the last input column of an input record, it is copied to that column of the output record.

. Period (concatenation character)
The period causes the value of the variable to be concatenated with the character string following the period when used at the end of a variable name. For example, if variable V has the value ABC:

"&V.DEF" yields "ABCDEF"

! Tab control character
The exclamation point serves as the default tab character for the)TB and the)TBA control statements. The file-tailoring tabbing function works like a typewriter tabbing operation. You can, however, specify in the)TB syntax that tabbing is not to take place if a tab stop is sensed at the same record position as the tab character.

< | > Conditional substitution control characters
The less-than, logical OR, and greater-than characters specify, respectively, the beginning, middle, and end of a conditional substitution string. For example:

<*string1*|*string2*>

where:

string1	Must contain at least one variable name
string2	Can be null

If the first variable in *string1* is not null, *string1* is substituted in the output record. If the first variable in *string1* is null, *string2* is substituted in the output record.

Example: An input skeleton contains:

```
)SET I = &Z
)SET J = VALUE_OF_J
)SET K = VALUE_OF_K
FIRST CONDITIONAL SUBSTITUTION RESULT: <&J|&K>
SECOND CONDITIONAL SUBSTITUTION RESULT: <&I|&J>
```

After processing, the file-tailoring output file contains:

```
FIRST CONDITIONAL SUBSTITUTION RESULT: VALUE_OF_J
SECOND CONDITIONAL SUBSTITUTION RESULT: VALUE_OF_J
```

Two consecutive control characters

Two consecutive control characters in the input record result in one control character being placed in the output record:

Characters	Yield	Comments
&&	&	
!!	!	
<<	<	
||	|	
>>	>	
..	.	Immediately following a variable name

The following characters can be overridden with the)DEFAULT control statement:

) & ? ! < | >

If any of these characters are overridden, the specified characters are substituted for those shown in the control character list on page 118.

Note: File tailoring treats an ampersand–blank combination in the input record as an invalid variable name.

File-tailoring control statements

A control statement must begin in column 1 with the control statement control character. The default control statement control character is a right parenthesis. Control statements cannot be continued on a second line.

Control words must be entered in uppercase.

General format of a control statement

)*control-word* [*parameter*]...]

where each parameter represents a name, value, operator, or keyword.

The parameters must be separated by one or more blanks, and cannot contain imbedded blanks. A parameter can be coded as:

- A character string
- A dialog variable name, preceded by an ampersand
- A concatenation of variable names and character strings

The current value of each variable is substituted before the control statement is evaluated. The rules for delimiting a variable name and for the use of ampersands, periods, double ampersands, and double periods are the same as for data records.

The following sections describe the specific control statements.

)BLANK — Insert blank lines

Use the)BLANK control statement to place blank lines in the output file at the point at which the)BLANK statement is encountered.

)BLANK control statement syntax

)BLANK [*number*]

number
: The specified number of blank lines are placed in the output file. If the number parameter is omitted, the default value 1 is used.

Example 1

Insert a single blank line in the output file.

```
)BLANK
```

Example 2

The number of blank lines inserted in the output file is equal to the current value of the variable SPACER.

```
)BLANK &SPACER
```

)CM — Comment

Use the)CM control statement to add commentary text to the skeleton.

)CM control statement syntax

)CM *comment*

The statement is treated as a comment. No tailoring is performed, and the record is not placed in the output file.

)DEFAULT — Define control characters

Use the)DEFAULT control statement to override the control characters.

The)DEFAULT statement takes effect immediately it is encountered. It remains in effect until the end of the skeleton is encountered, or until another)DEFAULT statement is encountered. It will be in effect only within the skeleton within which it occurs. It does not apply to any skeletons that are imbedded. If the)DEFAULT statement is used to change defaults within an imbed, it is in effect for only that imbed level. It does not apply to deeper or previous imbed levels.

)DEFAULT control statement syntax

)DEFAULT *abcdefg*

The characters represented by *abcdefg* override the default control characters), &, ?, !, <, |, and >, respectively. You must specify exactly seven characters.

a	Control statement control character
b	Variable name control character
c	Continuation record control character
d	Tab control character
efg	Beginning, middle, and end of conditional substitution control characters

Example 1

This example demonstrates that defaults changed using)DEFAULT do not take effect in imbedded skeletons.

An FTINCL (include skeleton) operation for skeleton SKEL1 changes the variable name control character, &, to the ¢ sign and then imbeds SKEL2.

```
Skeleton SKEL1          Skeleton SKEL2          File-tailoring output
)SKELETON               )SKELETON               A: USERNAME
)DEFAULT )¢?!<|>        AA: ¢A                  AA: ¢A
)SET A = USERNAME       AA: &A                  AA: USERNAME
A: ¢A                   )END                    A: USERNAME
)IM SKEL2
A: ¢A
)END
```

Example 2

This example demonstrates that defaults changed in an imbedded skeleton are not passed back to the skeleton performing the)IMBED.

Skeleton SKEL3 imbeds SKEL4, which changes the variable name control character & to the ¢ sign.

```
Skeleton SKEL3          Skeleton SKEL4          File-tailoring output
)SKELETON               )SKELETON               A: ¢A
)SET A = USERNAME       )DEFAULT )¢?!<|>        AA: USERNAME
A: ¢A                   AA: ¢A                  AA: &A
)IM SKEL4               AA: &A                  A: ¢A
A: ¢A                   )END
)END
```

)DOT and)ENDDOT — Iterate over table rows

Use the)DOT control statement to begin a group of records that will be processed iteratively. Use)ENDDOT to end this group.

The skeleton input records between the)DOT statement and the corresponding)ENDDOT statement are processed iteratively, once for each row in the named table, beginning with the first row. Any control statements can be used between the)DOT and the)ENDDOT control statements.

At the start of each iteration, the contents of the current table row are stored in the corresponding dialog variables. Those values can then be used as parameters in control statements or substituted in data records. If the table was already open, it remains open after file tailoring with the current row pointer (CRP) at position zero (TOP). If it was not open, it is opened automatically and then closed on completion of file tailoring.

Up to four levels of)DOT nesting are permitted. The same table cannot be processed recursively. The list of records must end with the)ENDDOT statement.

)DOT control statement syntax

)DOT *table-name*
  ⋮
)ENDDOT

table-name
The name of the table to be processed

Example

Take information from table ABC and write any blank table row as a blank line:

```
)DOT ABC
)SEL &LNAME=&Z
)BLANK 1
)ENDSEL
 &FNAME &LNAME
)ENDDOT
```

)END — End of file-tailoring skeleton

Use the)END control statement to end a file-tailoring skeleton definition.

Syntax

```
)END
```

SDF2/DM ignores any data that appears on lines following the)END statement.

)IM — Imbed skeleton

Use the)IM control statement to imbed another skeleton.

The specified skeleton is imbedded at the point at which the)IM statement is encountered.

Up to three levels of imbedding are permitted.

)IM control statement syntax

)IM *skel-name* [**NT**] [**OPT**]

skel-name	The name of the skeleton to be imbedded.
NT	The optional NT parameter indicates that no tailoring is to be performed on the imbedded skeleton. Since it causes the data to be imbedded as it is, without any control character or control statement processing, using the NT option improves performance.
OPT	The optional OPT parameter indicates that the skeleton is not required to be present in the skeleton library. If OPT is coded and the skeleton is not present, no error indication is given, and the record is ignored. If OPT is not coded and the skeleton is not present, a severe error occurs.

Example

The following example demonstrates how to use the NT parameter to prevent tailoring from occurring when imbedding a file. Using NT eliminates the need to change defaults in the imbedded skeleton when it contains default control characters.

An FTINCL of skeleton SKEL1 imbeds skeleton SKEL2 with the NT parameter.

Skeleton SKEL1	Skeleton SKEL2	File-tailoring output
`)SKELETON`	`)SKELETON`	`LBL1:`
`)SET A = LBL1`	`IF &A > 10 THEN`	`IF &A > 10 THEN`
`&A:`	`  &A = 0`	`  &A = 0`
`)IM SKEL2 NT`	`ELSE`	`ELSE`
`  GO TO &A`	`)END`	`  GO TO LBL1`
`)END`		

)SEL and)ENDSEL — Conditional section

Use the)SEL control statement to evaluate a relational expression for a true or false condition.

)SEL control statement syntax

)SEL *relational-expression*
⋮
)ENDSEL

If the condition is true, the skeleton input records between the)SEL and the corresponding)ENDSEL are processed. If the condition is false, these records are skipped. Up to eight levels of)SEL nesting are permitted. The list of records must end with an)ENDSEL statement.

Any other control statements can be used between the)SEL and the)ENDSEL control statements. For example, to write information from a table only if variable ABC is set to the name of that table, specify:

```
)SEL &ABC='TABNAME'
)DOT TABNAME
   &FNAME &LNAME
)ENDDOT
)ENDSEL
```

The relational expression consists of either a simple comparison of the form:

value1 operator value2

or of a combination of up to eight simple comparisons joined by connectors. The system variable Z can be used to represent a null or blank value.

The valid operators are:

EQ or =	NE or ¬=	
LT or <	NL or ¬<	GE or >=
GT or >	NG or ¬>	LE or <=

The valid connectors are | (OR) and && (AND). SDF2/DM evaluates connected expressions from left to right and evaluates the connectors with equal priority.

Examples

```
)SEL  &COND = YES
)SEL  &TEST1 not= &Z   |  &ABC = 5
)SEL  &TEST1 not= &Z  &&  &ABC = 5
```

)SET — Passing value to a variable

Use the)SET control statement to assign a value to a dialog variable.

)SET control statement syntax

)SET *variable* **=** *expression*

The *expression* can be specified as either:

literal

or

value1 operator value2

Where *literal* can be specified as:

string [**&***var-name*[**.***literal*]]

variable
: The name of the variable to be set.

  The variable name should not be preceded by an ampersand, unless the variable name is itself stored as a variable.

literal
: Any string without blanks or a variable name (preceded by an &) containing a literal.

value
: A numeric value or a variable name (preceded by an &) containing a numeric value.

operator
: A plus sign (+) or a minus sign (–).

A blank is required between the variable and the equal sign and between the equal sign and the expression.

To assign a null value to a dialog variable, use the system variable &Z.

Example

An input skeleton file contains:

```
)SET A = 1
)SET B = 2
)SET C = &A + &B
)SET C = &C + 5
)SET D = &Z
)SET E = ---&A.&B.&C.---
A is &A, B is &B, C is &C, D is &D, E is &E
```

The resulting output file contains:

```
A is 1, B is 2, C is 8, D is , E is ---128---
```

)SKELETON — File-tailoring skeleton definition

Use the)SKELETON statement to identify a file-tailoring skeleton definition.

Syntax

)SKELETON

The)SKELETON statement must be the first statement in the first line of the member.

)TB and)TBA — Set tab stops

Use the)TB control statement to set up to 16 tab stops. The default tab character, ! (exclamation point), tabs the output record to the next tab stop and fills the intervening spaces with blanks. The next character following an exclamation point in the input record is put at the tab stop location in the output record. A tab stop specifies a tab position in the output record.

standard tabbing If the tab stop value equals the current output position, the tabbing skips to the next tab stop value that is greater than the current output position. The input character following the tab character is then inserted at the position skipped to in the output record.

alternative tabbing If the tab stop value equals the current output position, the input character following the tab character is inserted at the current position in the output record.

You can thus write to the current position of the output record if a tab character in the input record is encountered at the same time as a tab stop is encountered in the output record.

)TB control statement syntax

)TB[**A**] *value1*[**A**] ... *value16*[**A**]

The way you specify alternative tabbing syntax on the)TB control statement determines whether only designated tab stop values or all tab stop values are affected, even if the tab stop value equals the current position in the output record when a tab character is encountered in the input record. If you specify:

```
)TB value1A ... value16A
```

only the tab stop values to which the character A is appended selectively cause tabbing to stop in any of those positions. If you specify:

```
)TBA  value1 ... value16
```

any tab stop value that equals the current position in the output record when a tab character is encountered in the input record causes tabbing to stop.

The character that you append for alternative tabbing must be an uppercase A. Appending an A to the)TB control word—)TBA—has the same effect as appending an A to all individual tab stop values. When you use the)TBA control word, appending an A to an individual tab stop value has no additional effect.

Example 1

This example uses the standard tabbing syntax.

Skeleton	After processing, the file-tailoring output record contains:	
`)TB 5 10 20`	Positions 1–4	contain blanks inserted by the first tab operation.
`!ABCDE!F`	Positions 5–9	Contain `ABCDE`. Standard tabbing occurs between `E` and `F` because tab stop 10 is at the same (not greater than) position of the output record at which the tab character is encountered in the input record.
	Positions 10–20	Contain blanks inserted by the second tab operation.
	Position 20	Contains `F`.

Example 2

This example uses alternative tabbing syntax for designated tab positions.

Skeleton	After processing, the file-tailoring output record contains:	
`)TB 5 10A 20`	Positions 1–4	Contain blanks inserted by the first tab operation.
`!ABCDE!F`	Positions 5–10	Contain `ABCDEF`. `F` immediately follows `E` because alternative tabbing is specified for tab position 10. This allows tabbing to stop in the current output record position (10) when the tab character is encountered in the input record.

Example 3

This example uses the alternative tabbing syntax for all tab positions.

Skeleton	After processing, the file-tailoring output record contains:	
`)TBA 3 6 10`	Positions 1–2	Contain the blanks inserted by the first tab operation.
`!ABC!DEF!GH`	Positions 3–5	Contain `ABC`. `D` immediately follows `C` because alternative tabbing is specified and a tab stop is set at the current output position (6).
	Positions 6–8	Contain `DEF`.
	Position 9	Contains a blank inserted by normal tabbing.
	Positions 10–11	Contain `GH`.

Chapter 8. Dialog management services

This chapter describes the dialog management services of SDF II. The services are listed in alphabetical order.

As appropriate for each service, the REXX command format and the PL/I call format are shown. If you are using a language other than REXX or PL/I, see “Invoking a service with a program call” on page 4 and the user’s guide that applies to your programming language.

ADDPOP — Start pop-up window mode

The ADDPOP service notifies the dialog manager that all subsequent panel displays are to appear in a pop-up window. No visible results appear on the screen until you issue a DISPLAY call.

When a pop-up window is displayed, the user must finish interacting with that pop-up window before continuing with the dialog in the underlying panel.

All subsequent panel displays will be in the pop-up window created with the ADDPOP call, until a REMPOP or another ADDPOP is called. Another ADDPOP call, after the first pop-up window has been displayed, creates a separate pop-up window.

Each pop-up window can have a window title embedded in the top of the window frame. If the title is longer than the window frame, the dialog manager truncates it. Set system variable ZWINTTL to the window title.

REXX command format

```
DGIEXEC ADDPOP [POPLOC(field-name)]
               [ROW(row)]
               [COLUMN(column)]
```

PL/I call format

```
CALL DGILINK ('ADDPOP  ' [, field-name
                         [, row
                         [, column]]]);
```

field-name
: Positions the pop-up window relative to the specified field in the currently displayed panel, rather than relative to the active window.

  If possible, the window is placed to the right of the field. If there is not enough room, the panel is placed to the left, above, or below the field. The window will not overlay the field unless this is the only way to display the window.

row
: Adjusts the window location by the specified number of rows, relative to the position that the window would otherwise have. A positive number moves the window down; a negative number moves the window up.

column
: Adjusts the window location by the specified number of columns, relative to the position that the window would otherwise have. A positive number moves the window right; a negative number moves the window left.

The position of the pop-up is determined as follows:

1. The default position is the top leftmost position of the underlying panel.
2. If *field-name* is specified, the window is repositioned so that it does not overlay the named field. The window is positioned to the right, to the left, above, or below the field.
3. If *row* is specified, the pop-up is moved down (or up for a negative number) the indicated number of rows.
4. If *column* is specified, the pop-up is moved right (or left for a negative number) the indicated number of columns.
5. If there is not enough room for the window at the specified location, SDF2/DM positions the window wherever it will fit.
6. A window can be repositioned on the screen by using the WINDOW command (see page 220).

Return codes

The following return codes are possible:

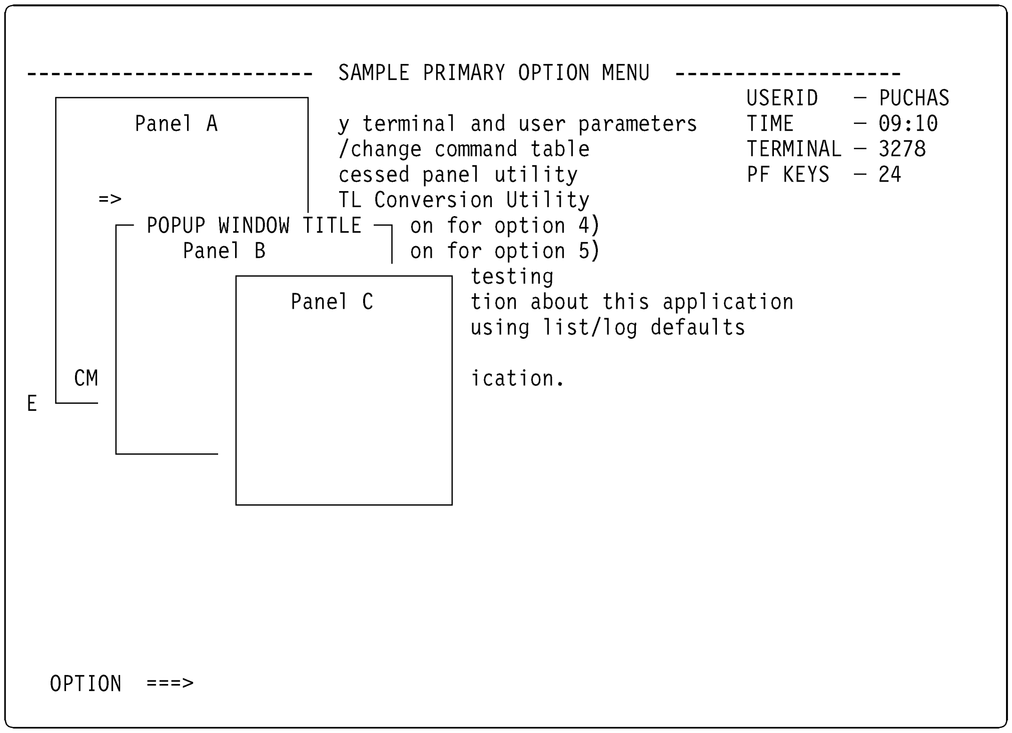
0	Normal completion.
12	An ADDPOP service call was issued either when no panel was displayed or when a pop-up window from a previous ADDPOP service call had not yet been displayed.
20	Severe error.

Example

The following procedure called after displaying a menu panel:

```
"DGIEXEC ADDPOP"
"DGIEXEC DISPLAY PANEL(A)"
"DGIEXEC ADDPOP POPLOC(SAM)"
ZWINTTL='POPUP WINDOW TITLE'
"DGIEXEC VPUT ZWINTTL"
"DGIEXEC DISPLAY PANEL(B)"
"DGIEXEC ADDPOP COLUMN(5)"
ZWINTTL=''
"DGIEXEC VPUT ZWINTTL'
"DGIEXEC DISPLAY PANEL(C)"
```

Results in:



```
------------------------  SAMPLE PRIMARY OPTION MENU  -------------------
                                                              USERID   - PUCHAS
  ┌────────────────────┐
  │     Panel A        │  y terminal and user parameters    TIME     - 09:10
  │                    │  /change command table             TERMINAL - 3278
  │                    │  cessed panel utility              PF KEYS  - 24
  │  =>                │  TL Conversion Utility
  │   ┌─ POPUP WINDOW TITLE ─┐ on for option 4)
  │   │     Panel B          │ on for option 5)
  │   │         ┌──────────────────┐ testing
  │   │         │    Panel C       │ tion about this application
  │   │         │                  │ using list/log defaults
  │   │         │                  │
  │ CM│         │                  │ ication.
E └───│         │                  │
      └─────────│                  │
                │                  │
                └──────────────────┘




  OPTION  ===>
```

CONTROL — Set processing modes

The CONTROL service either controls how the next DISPLAY call is processed or sets an error mode.

REXX command format

One of the following:

```
DGIEXEC CONTROL DISPLAY LOCK
DGIEXEC CONTROL DISPLAY LINE
DGIEXEC CONTROL DISPLAY REFRESH
DGIEXEC CONTROL DISPLAY SAVE
DGIEXEC CONTROL DISPLAY RESTORE
DGIEXEC CONTROL NONDISPL [ENTER  | END]
DGIEXEC CONTROL ERRORS [CANCEL | RETURN]
DGIEXEC CONTROL TEST {ON | OFF }
DGIEXEC CONTROL OPEN [APPLID(applid)]
DGIEXEC CONTROL CLOSE
DGIEXEC CONTROL CLEANUP
```

PL/I call format

One of the following:

```
CALL DGILINK ('CONTROL ', 'DISPLAY ', 'LOCK    ');
CALL DGILINK ('CONTROL ', 'DISPLAY ', 'LINE    ');
CALL DGILINK ('CONTROL ', 'DISPLAY ', 'REFRESH ');
CALL DGILINK ('CONTROL ', 'DISPLAY ', 'SAVE    ');
CALL DGILINK ('CONTROL ', 'DISPLAY ', 'RESTORE ');
CALL DGILINK ('CONTROL ', 'NONDISPL'[, 'ENTER   ' | 'END     ']);
CALL DGILINK ('CONTROL ', 'ERRORS  '[, 'CANCEL  ' | 'RETURN  ']);
CALL DGILINK ('CONTROL ', 'TEST    '{, 'ON      ' | 'OFF     ' });
CALL DGILINK ('CONTROL ', 'OPEN    '[, applid ]);
CALL DGILINK ('CONTROL ', 'OPEN    ');
CALL DGILINK ('CONTROL ', 'CLOSE   ');
CALL DGILINK ('CONTROL ', 'CLEANUP ');
```

DISPLAY LOCK
Specifies that the keyboard is to be locked for the next display. The next DISPLAY call is processed as if the user had pressed Enter. You can use CONTROL DISPLAY LOCK to display an “in process” message during a long-running operation.

DISPLAY LINE
Specifies that you expect line-mode input. When CONTROL DISPLAY LINE is processed, full-screen mode is turned off. The next DISPLAY rewrites the entire screen, and full-screen mode is turned on again.

You can also use CONTROL DISPLAY LINE to force full-screen mode to be turned on again after returning from a non-SDF2/DM application that turns off full-screen mode.

DISPLAY REFRESH
Specifies that you expect full-screen input from a non-SDF2/DM application. The next DISPLAY call erases the screen and then rewrites the entire screen.

CONTROL DISPLAY REFRESH does not force a return to full-screen mode. For this, you use CONTROL DISPLAY LINE.

DISPLAY SAVE
This service should be used in conjunction with the DISPLAY service. The state of the current screen including the current scroll position of a scrollable area or list is saved.

You can use the CONTROL DISPLAY SAVE and CONTROL DISPLAY RESTORE service to nest display services. After restoring the display, the panel previously saved can be redisplayed by specifying no panel name on the DISPLAY service.

CONTROL DISPLAY SAVE and CONTROL DISPLAY RESTORE must be issued in pairs.

DISPLAY RESTORE
Specifies the restoration of information previously saved by the CONTROL DISPLAY SAVE service. The screen image is restored exactly as it appeared when the SAVE was performed. Processing of the previous panel can then be resumed.

NONDISPL [**ENTER** | **END**]
Specifies that the next panel is not to be displayed. The next DISPLAY call is processed as if the user had pressed Enter (the default) or END. Only the)INIT and)PROC sections of the panel definition are processed.

ERRORS [**CANCEL** | **RETURN**]
Sets the error mode, which determines what happens when an error with a return code of 12 or higher is returned by a dialog management service.

CANCEL The panel shown in “Diagnostic information” on page 8 is to be displayed when a dialog management service returns a return code of 12 or higher. This is the default.

RETURN Control is to be returned to the dialog when a dialog management service returns a return code of 12 or higher. System variable ZERRLM contains the long message associated with the return code.

TEST {**ON** | **OFF**}
TEST ON disables the cache. The panel is always fetched from the library. TEST OFF is the default.

OPEN
Activates a new level of variables in the function pool. You cannot access any variable values that you previously set, until you close this level of variables (CONTROL CLOSE). Variables in the shared pool, including system variables, are not affected by CONTROL OPEN.

applid
: This parameter activates a profile data set. The member name of the profile is *userid*.PROF*xxxx*, where *xxxx* are the first four characters of the specified *applid*. System variable ZAPPLID contains the name of the application.

CLOSE
: Cancels a previous CONTROL OPEN statement.

CLEANUP
: Releases all memory allocated by SDF2/DM. It is especially important to call CLEANUP after an abend.

Return codes

The following return codes are possible:

0	Normal completion
20	Severe error

Example

Set the error processing mode to let the application prototype process return codes of 12 or higher.

```
"DGIEXEC CONTROL ERRORS RETURN"
```

or

```
CALL DGILINK ('CONTROL ','ERRORS  ','RETURN  ');
```

DISPLAY — Display panels and messages

The DISPLAY service displays a panel or redisplays the current panel, optionally with a message.

If you specify a panel name, the)INIT section of the panel definition is processed and the specified panel is displayed. If you do not specify a panel name, the)REINIT section of the current panel definition is processed and the current panel is redisplayed.

When the user enters a response to the panel, the values of the input fields are stored in dialog variables, and the)PROC section of the panel definition is processed.

REXX command format

```
DGIEXEC DISPLAY [PANEL(panel-name)]
                [MSG(message-id)]
                [CURSOR(cursor-field-name)]          .
                [CSRPOS(cursor-position)]
                [MSGLOC(message-field-name)]
                [CSRIDX(cursor-index)]
```

PL/I call format

```
CALL DGILINK ('DISPLAY '[, panel-name
                        [, message-id
                        [, cursor-field-name
                        [, cursor-position
                        [, message-field-name
                        [, cursor-index]]]]]]);
```

panel-name
: The name of the panel. If you do not specify a panel name, the current panel is redisplayed.

message-id
: The ID of a message to be displayed on the panel. This value overrides any message ID that you specified on a SETMSG statement. If the panel definition sets a value of .MSG either directly or through the TRANS or VER statement, this value overrides the message ID that you specify on the DISPLAY statement.

cursor-field-name
: The name of the field where the cursor is positioned. If the panel definition sets a value of .CURSOR, this value overrides the cursor field name that you specify on the DISPLAY statement.

  For information about the default cursor field, see the description of .CURSOR in Chapter 4, “Control variables” on page 100.

cursor-position
The position of the cursor within the cursor field. If the panel definition sets a value of .CSRPOS, this value overrides the cursor position that you specify on the DISPLAY statement. The default is 1.

message-field-name
Positions the message pop-up relative to the named field. By default, the pop-up appears at the bottom of the screen or below the active pop-up window if one exists.

For upward compatibility, this parameter should be specified only if the message will be displayed in a pop-up window.

cursor-index
An index indicating the row in the field specified on the CURSOR parameter in which to place the cursor. This parameter applies only to fields that display array variable data (for example, in a scrollable list).

Return codes

The following return codes are possible:

0	Normal completion.
8	User requested termination of any panel by using the END or RETURN command, or requested termination of a CUA panel by using the EXIT or CANCEL command.
12	The specified panel, message, message location field, or cursor field could not be found.
16	Truncation or translation error in storing defined variables.
20	Severe error.

Example

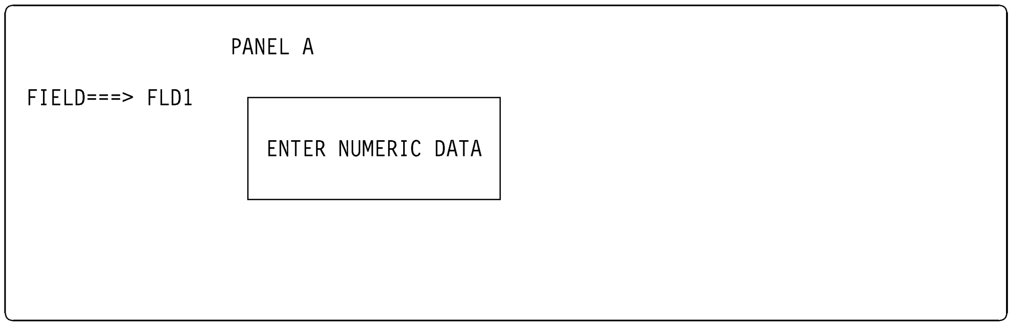
The following message is contained in a message member:

```
TSTA110              .WINDOW=RESP
'ENTER NUMERIC DATA'
```

The following DISPLAY call:

```
"DGIEXEC DISPLAY PANEL(A) MSG(TSTA110) MSGLOC(FLD1)"
```

results in:



```
                 PANEL A

FIELD===> FLD1
                   ENTER NUMERIC DATA
```

FTCLOSE — End file tailoring

The FTCLOSE service is used to terminate the file-tailoring process and to indicate the final disposition of the file-tailoring output.

Specify a *member-name* parameter if the output is to be stored in a particular library. The file-tailoring output is given the specified member name. No error condition results if the *member-name* parameter is not specified and the output is stored in the work library DGIWORK.DGI.

The NOREPL parameter specifies that an existing member in the file-tailoring output library is not to be overlaid by the current FTCLOSE service. If a member of the same name already exists, the FTCLOSE service request is terminated with a return code of 4 and the original member remains unaltered.

The dialog manager sets the system variables ZTEMPF to *member.type* and ZTEMPN to *lib.sublib*, which is set to DGIWORK.DGI. These variables are available to the caller for locating the file-tailoring output.

REXX command format

```
DGIEXEC FTCLOSE [NAME(member-name)]
                [LIBRARY(lib.sublib)]
                [NOREPL]
                [TYPE(member-type)]
```

PL/I call format

```
CALL DGILINK ('FTCLOSE ' [, member-name
                         [, lib.sublib
                         [, 'NOREPL '
                         [, member-type]]]]);
```

member-name
Specifies the name of the member in the output library that is to contain the file-tailoring output. Include this parameter if you specify an output library.

lib.sublib
Specifies the output library that is to contain the file-tailoring output. If you specify this parameter, specify also the member name.

Specify either an output library on the FTCLOSE statement or TEMP on the FTOPEN statement. An output library parameter on the FTCLOSE statement overrides a TEMP parameter on the FTOPEN statement.

NOREPL
Specifies that FTCLOSE is not to overlay an existing member in the output library.

member-type
Specifies the type of the member in the output library that is to contain the file-tailoring output.

Return codes

The following return codes are possible:

0	Normal completion.
4	Member already exists in the output library and NOREPL was specified. The original member is unchanged.
8	File tailoring not active. FTOPEN was not used prior to FTCLOSE.
16	Output library could not be opened.
20	Severe error.

Example

End the file-tailoring process and store the result of the processing in the file-tailoring output library in member TELOUT.

```
"DGIEXEC FTCLOSE NAME(TELOUT)"
```

or

```
CALL DGILINK ('FTCLOSE ','TELOUT ');
```

FTERASE — Erase file-tailoring output

The FTERASE service erases a file-tailoring output member in a library.

REXX command format

```
DGIEXEC FTERASE member-name [LIBRARY(lib.sublib)]
                            [TYPE(member-type)]
```

PL/I call format

```
CALL DGILINK ('FTERASE ', member-name[, lib.sublib
                                      [, member-type]]);
```

member-name
: Specifies the name of the member that is to be deleted from the output library.

lib.sublib
: Specifies the library from which the file-tailoring output member is to be deleted.

member-type
: Specifies the member type in the library from which the file-tailoring output member is to be deleted.

Return codes

The following return codes are possible:

0	Normal completion
8	File does not exist
16	Output library could not be opened
20	Severe error.

Example

Erase member TELOUT in the file-tailoring output library.

```
"DGIEXEC FTERASE TELOUT"
```

or

```
CALL DGILINK ('FTERASE ','TELOUT ');
```

FTINCL — Include a skeleton

The FTINCL service specifies the skeleton that is to be used to produce the file-tailoring output. If an FTOPEN service has not already been issued, the FTINCL service performs the equivalent of an FTOPEN, without the TEMP keyword, before processing the specified skeleton.

REXX command format

DGIEXEC FTINCL *skel-name* [**NOFT**]

PL/I call format

CALL DGILINK ('FTINCL ', *skel-name*[**, 'NOFT '**]**);**

skel-name
Specifies the name of the skeleton.

NOFT
Specifies that no file tailoring is to be performed on the skeleton: the entire skeleton is to be copied unchanged to the output file, with no variable substitution or interpretation of control records.

Return codes

The following return codes are possible:

0	Normal completion
8	Skeleton does not exist
16	Data truncated or output library could not be opened
20	Severe error

Example

Perform file tailoring using the file-tailoring skeleton named TELSKEL, which is a member in the file-tailoring skeleton library, to control processing.

```
"DGIEXEC FTINCL TELSKEL"
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL DGILINK ('FTINCL ','TELSKEL ');
```

FTOPEN — Begin file tailoring

The FTOPEN service, which begins the file-tailoring process, allows skeleton files to be accessed from the SDF2/DM object library.

REXX command format

DGIEXEC FTOPEN [**TEMP**]

PL/I call format

CALL DGILINK ('FTOPEN '[, **'TEMP '**]**);**

TEMP
Specifies that the output of the file-tailoring process is to be placed in a temporary member of the library DGIWORK.DGI.

Specify either TEMP on the FTOPEN statement or an output library on the FTCLOSE statement. A TEMP parameter on the FTOPEN statement is overridden by an output library parameter on the FTCLOSE statement.

Return codes

The following return codes are possible:

0	Normal completion
8	File tailoring already in progress
16	Skeleton or output library could not be opened
20	Severe error

Example

Prepare for access (open) both the file-tailoring skeleton and file-tailoring output libraries.

```
"DGIEXEC FTOPEN"
```

or

```
CALL DGILINK ('FTOPEN ');
```

GETMSG — Get a message

The GETMSG service verifies that a message exists, and gets the information about the message that you request. The message is retrieved and variable substitution is performed when the GETMSG call is processed.

If the message exists, the long message and alarm always have a value. If no ALARM value is specified for the message, the default value NO is used.

The other parameters may or may not be specified. If a value is not specified, GETMSG returns a null value.

All the parameters except *message-id* are optional. If no additional parameters are specified, GETMSG simply checks that the message exists.

REXX command format

```
DGIEXEC GETMSG MSG(message-id)  [SHORTMSG(short-message-var)]
                                [LONGMSG(long-message-var)]
                                [ALARM(alarm-var)]
                                [HELP(help-var)]
                                [TYPE(type-var)]
                                [WINDOW(window-var)]
```

PL/I call format

```
CALL DGILINK ('GETMSG  ', message-id [, short-message-var
                                     [, long-message-var
                                     [, alarm-var
                                     [, help-var
                                     [, type-var
                                     [, window-var]]]]]]);
```

message-id
: The message ID for which information is to be retrieved.

short-message-var
: The variable in which the short message text, if any, is stored.

long-message-var
: The variable in which the long message text is stored.

alarm-var
: The variable in which the alarm indicator (YES or NO) is stored. If a value is specified for .TYPE in the message member, the type value determines the alarm value. Otherwise, the value of .ALARM (with a default of NO) is used.

help-var
: The variable in which the help panel name, if any, is stored. The help panel name is specified in the .HELP keyword.

type-var
The variable in which the type value, if any, is stored. The type value (NOTIFY, WARNING, ACTION, or CRITICAL) is specified in the .TYPE keyword.

window-var
The variable in which the window value, if any, is stored. If .TYPE=CRITICAL is specified in the message member, the window type is RESP. Otherwise, the value of .WINDOW is used.

Return codes

The following return codes are possible:

0	Normal completion
12	The specified message could not be found
20	Severe error

Example

For the message named ABCS102, return the text of the long message in variable ERRMSG and the help panel name in variable HPANEL:

```
"DGIEXEC GETMSG MSG(ABCS102) LONGMSG(ERRMSG) HELP(HPANEL)"
```

or

```
CALL DGILINK ('GETMSG  ','ABCS102 ',' ','ERRMSG  ',' ','HPANEL  ');
```

The HELP service

The HELP service displays a help panel. It must be preceded by a DISPLAY service call. The help panel is displayed in a pop-up window if the WINDOW keyword is specified within the help panel.

REXX command format

```
DGIEXEC HELP [PANEL(help-panel-name)]
             [FIELD(field-name)]
             [FLDPOS(field-position)]
             [FLDLEN(field-length)]
```

PL/I call format

```
CALL DGILINK ('HELP    ' [, help-panel-name
                          [, field-name
                          [, field-position
                          [, field-length]]]]);
```

help-panel-name
The name of the panel to be displayed. If no panel name is specified, the help-for-help panel (DGIRHELP) is displayed.

field-name
Positions the help pop-up window relative to the specified field. By default, the field where the cursor was positioned (the value of the .CURSOR control variable) is used.

field-position
The start position of a logical field within the specified field. The default is 1.

field-length
The length of a logical field within the specified field. If a *field-position* value is specified, the default 1 is used. Otherwise, the default is the length of the field.

For most fields, you would specify only a field name. If you want to define field help for a logical field within a dynamic area, specify values for *field-position* and *field-length* because the entire dynamic area is considered one field. The help pop-up is then placed relative to this logical field.

Return codes

The following return codes are possible:

8	The user terminated help
16	Truncation error
20	Severe error

Example 1

The following procedure displays help for the command line. The position of the help pop-up depends on the cursor position.

```
"DGIEXEC HELP PANEL(DGIRZCMD)"
```

Example 2

The following procedure displays field help for a field within a dynamic area:

```
"DGIEXEC HELP PANEL(DGIFDYNA) FIELD(DYN) FLDPOS(150) FLDLEN(5)"
```

The entire dynamic area is considered one field, with field name DYN. A logical field within this dynamic area begins 150 spaces into the dynamic area and extends for 5 spaces. Counting begins with the upper left-hand corner of the dynamic area and continues from left to right and top to bottom. The help panel will be displayed relative to this logical field.

LIBDEF — Allocate application prototype libraries

The LIBDEF service identifies an additional library that can contain panels, keylists, and messages. Only one library, in addition to the default search chain (`// LIBDEF SOURCE,SEARCH=`), can be used at any time. Each LIBDEF call replaces the library specified on the previous LIBDEF call.

The library that you specify using LIBDEF is searched for panels, keylists, messages and so on before SDF2/DM searches the libraries specified in the VSE system control statement:

```
// LIBDEF SOURCE,SEARCH=...
```

If your application prototype is available in several languages, you could put the English version in one library, the Spanish version in another, and so on. Your application prototype would then use the LIBDEF service, with a variable for the library name, to select the appropriate language library.

REXX command format

DGIEXEC LIBDEF DGIPLIB [**ID(***libname***)**]

PL/I call format

CALL DGILINK ('LIBDEF ', 'DGIPLIB '[, *libname*]**);**

libname
: The optional *lib.sublib* name and the optional member *type*. The default member type is A. The default libraries are those specified in the VSE system control statement:

```
// LIBDEF SOURCE,SEARCH=...
```

The specification must be enclosed in parenthesis. For example:

```
 (DGILIB.DGIPGER A)
```

If you do not specify a library name, the previous LIBDEF setting is cleared and only the default search chain is used.

Return codes

The LIBDEF service return codes are:

0	Normal completion
16	Library does not exist
20	Severe error

Example

If the library DGILIB.DGIPANLS contain the panels and a member type of PANEL is used, LIBDEF is used as follows:

```
  "DGIEXEC LIBDEF DGIPLIB ID('DGILIB.DGIPANLS PANEL')"
```

or

```
  CALL DGILINK('LIBDEF  ', 'DGIPLIB ', '(DGILIB.DGIPANLS PANEL)');
```

LINEMSG — Display a message in line mode

The LINEMSG service displays a message in line mode. You can use it for debugging your application prototype or for displaying error messages.

REXX command format

DGIEXEC LINEMSG *message-text*

PL/I call format

CALL DGILINK ('LINEMSG ', *message-text*, *length***);**

message-text
Specifies the text to be displayed.

length
Specifies the length of the message text.

Return codes

0	Normal completion
20	Severe error

Example

```
"DGIEXEC LINEMSG At the top of the display loop..."
```

or

```
CALL DGILINK('LINEMSG ','At the top of the display loop...',33);
```

PQUERY — Obtain panel information

The PQUERY service verifies that a dynamic area exists, and gets the information about the dynamic area that you request. Variable substitution is performed when the PQUERY call is processed.

All the parameters except the panel name and dynamic-area name are optional. If no additional parameters are specified, PQUERY simply checks that the dynamic area exists.

REXX command format

```
DGIEXEC PQUERY  PANEL(panel-name)
                AREANAME(dynamic-area-name)
                [AREATYPE(area-type-var)]
                [WIDTH(width-var)]
                [DEPTH(depth-var)]
                [ROW(row-number-var)]
                [COLUMN(column-number-var)]
```

PL/I call format

```
CALL DGILINK ('PQUERY  ' , panel-name
                         , dynamic-area-name
                         [, area-type-var
                         [, width-var
                         [, depth-var
                         [, row-number-var
                         [, column-number-var]]]]]);
```

panel-name
: The name of the panel that contains the dynamic area.

dynamic-area-name
: The name of the dynamic area.

area-type-var
: The variable in which the area type is to be stored. If the return code is zero, DYNAMIC is returned left-justified and padded with blanks.

width-var
: The variable in which the number of columns in the dynamic area is to be stored. For a call, the variable should be defined as a fullword fixed integer.

depth-var
: The variable in which the number of rows in the dynamic area is to be stored. If EXTEND(ON) was specified for the dynamic area, this is the number of rows after the panel body is expanded to fill the screen. For a call, the variable should be defined as a fullword fixed integer.

row-number-var
: The variable in which the row number of the top left-hand position of the dynamic area is to be stored. For a call, the variable should be defined as a fullword fixed integer.

column-number-var
The variable in which the column number of the top left-hand position of the dynamic area is to be stored. For a call, the variable should be defined as a fullword fixed integer.

Return codes

The following return codes are possible:

0	Normal completion.
8	The dynamic area does not exist in the panel.
12	The panel does not exist.
16	Not all values are returned, because insufficient space was provided.
20	Severe error.

Example

The following procedure finds the depth of dynamic area DYNA1 on panel XYZ, and puts the value in variable DYNDEP:

```
"DGIEXEC PQUERY PANEL(XYZ) AREANAME(DYNA1) DEPTH(DYNDEP)"
```

REMPOP — Remove a pop-up window

The REMPOP service removes the current pop-up window. Any DISPLAY call will then display a panel in the full panel area, or in a higher level pop-up window if one is active.

Between an ADDPOP call and a REMPOP call, any DISPLAY call displays a panel in a pop-up window.

REXX command format

```
DGIEXEC REMPOP [ALL]
```

PL/I call format

```
CALL DGILINK ('REMPOP  '[, 'ALL     ']);
```

ALL
Removes all pop-up windows. If you do not specify ALL, only one pop-up window is removed.

Return codes

The following return codes are possible:

0	Normal completion.
16	A pop-up window does not exist.
20	Severe error.

SELECT — Select a panel or function

The SELECT service can be used to display a hierarchy of selection panels or to invoke a function.

Within a dialog function, a program can invoke another program, using a standard CALL or LINK convention. However, when the invoked program is a new dialog function, the SELECT service should be used to invoke the new function. One of the following:

REXX command format

```
DGIEXEC SELECT PANEL(panel-name)  [OPT(option)]
                                  [NEWAPPL(applid)]
                                  [POPUP]
                                  [POPLOC(field-name)]
                                  [ROW(row)]
                                  [COLUMN(column)]

DGIEXEC SELECT PGM(program-name)  [PARM(parameters)]
                                  [NEWFUNC]
                                  [NEWAPPL(applid)]

DGIEXEC SELECT CMD(command)       [NEWAPPL(applid)]
```

PL/I call format

```
CALL DGILINK ('SELECT  ', length, keywords[, userparm]);
```

panel-name
The name of the selection panel to be displayed.

option
Specifies an initial option, which must be a valid option on the menu specified in *panel-name*. Specifying an option causes direct entry to that option without displaying the menu. The menu is processed in nondisplay mode, as though the user had entered the option.

NEWFUNC
Activates a new level of variables in the function pool. The same function as CONTROL OPEN is performed.

applid
Activates a new level of variables in the function and profile pool. The same function as CONTROL OPEN APPLID(*applid*) is performed.

POPUP
Specifies that the selection panel specified in *panel-name* is to be displayed in a pop-up window.

field-name
Positions the pop-up window relative to the specified field in the currently displayed panel, rather than relative to the active window.

If possible, the window is placed to the right of the field. If there is not enough room, the panel is placed to the left, above, or below the field. The window will not overlay the field unless this is the only way that the window could be displayed.

row
Adjusts the window location by the specified number of rows, relative to the position that the window would otherwise have. A positive number moves the window down; a negative number moves the window up.

column
Adjusts the window location by the specified number of columns, relative to the position that the window would otherwise have. A positive number moves the window right; a negative number moves the window left.

program-name
Specifies the name of a program that is to be invoked as a dialog function. If the program is coded in PL/I, it must be a MAIN procedure.

program-name must specify a name of a PHASE loadable using the CDLOAD macro.

parameters
Specifies input parameters to be passed to the program. The parameters within the parentheses are passed as a single character string, preceded by a halfword containing the length of the character string, in binary. The length value does not include itself.

Parameters passed from the SELECT service to a PL/I program can be declared on the procedure statement in the standard way:

```
XXX: PROC(PARM) OPTIONS(MAIN);
     DCL PARM CHAR(nnn) VAR;
```

command
Specifies a REXX procedure name.

length
Specifies the length of the buffer containing the selection keywords. This parameter must be a fullword fixed binary number.

keywords
Specifies the name of the buffer containing the selection keywords. This is a character string parameter. The selection keywords in the buffer are specified in the same form as they would be coded for the DGIEXEC command. For example:

```
 SELCMD='PANEL(DGISMAIN) OPT(1.2) NEWAPPL(TEST)'
```

userparm
Specifies a fullword saved by SDF2/DM and passed to a selected program in register zero and as a second parameter in addition to the PARM specification. It can be used to pass, for example, a global control block address across selected functions.

The PL/I declaration to access the user parameter, passed from the SELECT service, could look like this:

```
XXX: PROC(PARM, UPARM) OPTIONS(MAIN);
     DCL PARM  CHAR(nnn) VAR,
         UPARM CHAR(4);
```

Return codes

The following return codes are possible:

0	Normal completion.
4	Normal completion. The RETURN command was entered or the EXIT option was specified from the selected menu or from some lower-level menu.
12	The panel does not exist, or the program or command was not found.
16	Truncation or translation error in storing defined variables.
20	Severe error.

The return code returned by the selected program or function is held in the system variable ZSELRC.

Example

The following procedure displays a menu panel.

```
"DGIEXEC SELECT PANEL(DGISMAIN)"
```

SETMSG — Set next message

The SETMSG service specifies which message will be displayed on the next panel. If CONTROL NONDISPL is in effect when the next DISPLAY call is processed, the message is not displayed but is saved until the first panel without CONTROL NONDISPL.

Only one message at a time can be displayed on a panel. If several possible messages could be displayed, the following sequence is used:

1. The message specified on the first SETMSG statement.
2. Any message specified on a subsequent SETMSG statement overrides the previous SETMSG unless you specify the COND keyword.
3. Any message specified on the DISPLAY statement overrides SETMSG.
4. Any message specified in the)INIT section of the panel definition overrides DISPLAY.

When the SETMSG call is processed, the message is retrieved and variable substitution is performed. If the user requests help for the message, variables in the help panel are substituted when HELP is requested and thus may have different values than when the SETMSG call was processed.

REXX command format

```
DGIEXEC SETMSG MSG(message-id)  [COND]
                                [MSGLOC(field-name)]
```

PL/I call format

```
CALL DGILINK ('SETMSG  ', message-id [, 'COND    '
                                        [, field-name]]);
```

message-id
The message ID that is to be displayed on the next panel.

COND
Specifies that the message is displayed only if no previous SETMSG request is pending.

field-name
Positions the message pop-up relative to the named field. By default, the pop-up appears at the bottom of the screen or below the active pop-up window if one exists.

For upward compatibility, specify this parameter only if the message will be displayed in a pop-up window.

Return codes

The following return codes are possible:

0	Normal completion.
4	A SETMSG call with the COND parameter was issued and a SETMSG request was pending.
12	The specified message field name or message was not found.
20	Severe error.

Example 1

On the next panel that is displayed, put message ABCX015:

```
"DGIEXEC SETMSG MSG(ABCX015)"
```

or

```
CALL DGILINK ('SETMSG  ','ABCX015 ');
```

Example 2

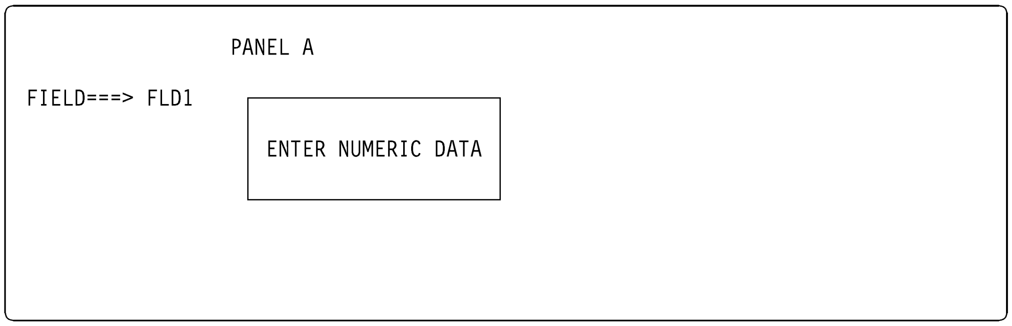
The following message is contained in a message member:

```
TSTA110               .WINDOW=RESP
'ENTER NUMERIC DATA'
```

The following SETMSG and DISPLAY calls:

```
"DGIEXEC SETMSG MSG(TSTA110) MSGLOC(FLD1)"
"DGIEXEC DISPLAY PANEL(A)"
```

result in:



```
                  PANEL A

FIELD===> FLD1
                    ENTER NUMERIC DATA
```

TBADD — Add a row to a table

The TBADD service adds a new row of variables to a table. The new row is added either immediately following the current row, pointed to by the current row pointer (CRP), or is added at a point appropriate for maintaining the table in the sequence specified in a previously processed TBSORT request. The CRP is set to point to the newly inserted row.

The current contents of all dialog variables that correspond to columns in the table, which were specified by the KEYS and NAMES parameters in a TBCREATE, are saved in the row.

Additional variables—those not specified when the table was created—can also be saved in the row. These extension variables apply only to this row, not to the entire table. The next time the row is updated, the extension variables must be respecified if they are to be rewritten.

For tables with keys, the table is searched to ensure that the new row has a unique key. The current contents of the key variables, which are dialog variables that correspond to keys in the table, are used as the search argument.

For tables without keys, no duplicate checking is performed.

REXX command format

DGIEXEC TBADD *table-name* [**SAVE(***name-list***)**][**ORDER**]

PL/I call format

CALL DGILINK ('TBADD   ', *table-name*[**,** *name-list*[**, 'ORDER   '**]]**);**

table-name
: Specifies the name of the table to be updated.

name-list
: Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created.

ORDER
: Specifies that the new row is to be added to the table in the order specified in the sort information record. A TBSORT must have been performed for this table prior to use of this keyword. For tables with keys, the table is searched to ensure that the new row has a unique key. If a row with the same key already exists, the row is not added. This keyword is ignored if the table has never been sorted. If this keyword is omitted, any existing sort information record is nullified; to restore the record, another TBSORT is required.

  When a newly inserted row has sort field names equal to the sort field names of an existing row, the insertion is made after the existing row.

Return codes

The following return codes are possible:

0	Normal completion.
4	The *number-of-rows* parameter was specified but storage was obtained for only a single row.
8	A row with the same key already exists; the CRP is set to TOP (zero). Returned only for tables with keys.
12	Table is not open.
16	Numeric conversion error; see numeric restrictions for TBSORT on page 197. Returned only for sorted tables.
20	Severe error.

Example

Add a row to the table TELBOOK, based on the sort information record, copying to the row values from function pool variables whose names match those of table variables.

```
"DGIEXEC TBADD TELBOOK ORDER"
```

or

```
CALL DGILINK ('TBADD   ','TELBOOK ',' ','ORDER   ');
```

TBBOTTOM — Set the row pointer to bottom

The TBBOTTOM service sets the current row pointer (CRP) to the last row of a table and retrieves the row unless the NOREAD parameter is specified.

If NOREAD is not specified, all variables in the row, including any key, name, and extension variables, are stored in the corresponding dialog variables. A list of extension variable names can also be retrieved.

REXX command format

```
DGIEXEC TBBOTTOM table-name [SAVENAME(var-name)]
                            [ROWID(rowid-name)]
                            [NOREAD| SHARED]
                            [POSITION(crp-name)]
```

PL/I call format

```
CALL DGILINK ('TBBOTTOM', table-name[, var-name
                                    [, rowid-name
                                    [, 'NOREAD ' |'SHARED '
                                    [, crp-name]]]]);
```

table-name
Specifies the name of the table to be used.

var-name
Specifies the name of a variable in which a list of extension variable names contained in the row is to be stored. The list is enclosed in parentheses, and the names within the list are separated by a blank.

rowid-name
Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. Later, this identifier can be specified in the ROW parameter of TBSKIP to cause the CRP to point to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

NOREAD
Specifies that the variables contained in the requested row are not to be read into the variable pool.

SHARED
Specifies that the variables contained in the requested row are to be read into the shared pool.

crp-name
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returned is zero.

Return codes

The following return codes are possible:

0	Normal completion
8	Table is empty; the CRP is set to TOP (zero)
12	Table is not open
16	Variable value has been truncated or insufficient space was provided to return all extension variable names
20	Severe error

Example

Move the current row pointer (CRP) of the table TELBOOK to the last row of the table. From this row, store variable values in the function pool variables whose names match those of the table variables.

```
"DGIEXEC TBBOTTOM TELBOOK"
```

or

```
CALL DGILINK ('TBBOTTOM','TELBOOK ');
```

TBCLOSE — Close and save a table

The TBCLOSE service terminates processing of the specified table and deletes the virtual storage copy, which is then no longer available for processing.

If the table was opened in write mode, TBCLOSE copies the table from virtual storage to the table output library. When storing a table in a table library, the user can give it a new name.

If the table was opened in nowrite mode, TBCLOSE simply deletes the virtual storage copy.

A TBCLOSE request for a shared table causes the use count in the table to be decremented by one. If the use count is zero, the TBCLOSE service is performed. If the count is not zero, a TBSAVE service is performed. This leaves the table available for continued processing, as long as the use count is greater than zero.

REXX command format

```
DGIEXEC TBCLOSE table-name [NAME(alt-name)]
                           [LIBRARY(lib.sublib)]
                           [TYPE(table-type)]
```

PL/I call format

```
CALL DGILINK ('TBCLOSE ', table-name[, any
                                   [, alt-name
                                   [, any
                                   [, lib.sublib
                                   [, table-type]]]]);
```

table-name
: Specifies the name of the table to be closed.

any
: Reserved.

alt-name
: Specifies an alternative name for the table. The table is stored with the alternative name in the table library. Any table with that name in the table library is replaced. If the table being saved exists with the original name in the table library that copy remains unchanged.

lib.sublib
: Specifies the library that is to contain the table.

table-type
: Specifies the type of the member in the library that is to contain the table.

Return codes

The following return codes are possible:

0	Normal completion
12	Table is not open
16	Table library could not be opened
20	Severe error

Example

Close the table TELBOOK.

```
"DGIEXEC TBCLOSE TELBOOK"
```

or

```
CALL DGILINK ('TBCLOSE ','TELBOOK ');
```

TBCREATE — Create a new table

The TBCREATE service creates a new table in virtual storage and prepares it for processing.

TBCREATE allows specification of the variable names that correspond to columns in the table. These variables will be stored in each row of the table. Additional extension variables can be specified for a particular row when the row is written to the table.

One or more variables can be defined as keys for accessing the table. If no keys are defined, only the current row pointer can be used for update operations.

REXX command format

```
DGIEXEC TBCREATE table-name [KEYS(key-name-list)]
                            [NAMES(name-list)]
                            [WRITE | NOWRITE]
                            [REPLACE]
                            [TYPE(table-type)]
                            [DESCRIPTION(description)]
```

PL/I call format

```
CALL DGILINK ('TBCREATE', table-name[, key-name-list
                                    [, name-list
                                    [, 'WRITE   ' | 'NOWRITE '
                                    [, 'REPLACE '
                                    [, any
                                    [, any
                                    [, table-type
                                    [, description]]]]]]]);
```

table-name
: Specifies the name of the table to be created. The name can be from 1–8 alphanumeric characters in length, beginning with an alphabetic character.

key-name-list
: Specifies the variables, by name, that are to be used as keys for accessing the table. See the discussion of name lists under “Parameter types for the DGILINK call” on page 6 for the specification of name lists. If this parameter is omitted, the table will not be accessible by keys.

name-list
: Specifies the nonkey variables, by name, to be stored in each row of the table.

  If *key-name-list* and *name-list* are omitted, the table can contain only extension variables, which must be specified when a row is written to the table.

WRITE
Specifies that the table is permanent—it is to be written to disk by the TBSAVE or TBCLOSE service. The disk copy is not actually created until the TBSAVE or TBCLOSE service is invoked.

The WRITE/NOWRITE usage of a shared table must be consistent on all TBCREATE and TBOPEN requests. That is, all requests for a given shared table that result in concurrent use of that table must specify the same WRITE or NOWRITE attribute.

NOWRITE
Specifies that the table is for temporary use only. When processing is complete, a temporary table should be deleted by the TBEND or TBCLOSE service.

REPLACE
Specifies that an existing table is to be replaced. If a table of the same name is currently open, it is deleted from virtual storage before the new table is created, and return code 4 is issued. If the WRITE parameter is also specified and a duplicate table name exists in the table input library, the table is created and return code 4 is issued. The duplicate table is not deleted from the input library. However, if TBSAVE or TBCLOSE is issued for the table, the existing table is replaced with the current table.

A table currently existing in virtual storage in shared mode cannot be replaced. If this is attempted, a return code of 8 results. Further, a shared table cannot be replaced by a nonshared table, and vice versa.

any
Reserved.

table-type
Specifies the type of the member in the library that is to contain the table.

description
A character string of up to 32 characters that describes the table.

Return codes

The following return codes are possible:

0	Normal completion.
4	Normal completion—a duplicate table exists but REPLACE was specified.
8	The table already exists and REPLACE was not specified.
12	Table in use.
20	Severe error.

Example 1

In a REXX EXEC, create a permanent table, TELBOOK, to contain the variable TABKEY and other variables, the names of which are specified in dialog variable TABVARS. The key field is TABKEY.

```
"DGIEXEC TBCREATE TELBOOK KEYS(TABKEY) NAMES("TABVARS")"
```

Example 2

In a PL/I program, create a permanent table, TELBOOK, to contain the variable TABKEY and other variables, the names of which are specified in the program variable TABVARS. The variable TABVARS has been made accessible to SDF2/DM by a previous VDEFINE operation. The key field is TABKEY.

```
CALL DGILINK ('TBCREATE','TELBOOK ','TABKEY  ',TABVARS);
```

Example 3

In a PL/I program, create a permanent nonkeyed table, NKTBL, where FNAME, LNAME, PHONE, and LOC are the nonkey table variables.

```
CALL DGILINK ('TBCREATE','NKTBL ',' ',
            '(FNAME LNAME PHONE LOC)');
```

TBDELETE — Delete a row from a table

The TBDELETE service deletes a row from a table.

For tables with keys, the table is searched for the row to be deleted. The current contents of the key variables, which are dialog variables that correspond to keys in the table, are used as the search argument. If the table has no keys, the row is determined by the position of the current row pointer (CRP).

For tables without keys, the row pointed to by the CRP is deleted.

The CRP is always updated to point to the row prior to the one that was deleted.

REXX command format

DGIEXEC TBDELETE *table-name*

PL/I call format

CALL DGILINK ('TBDELETE', *table-name***);**

table-name
  Specifies the name of the table from which the row is to be deleted.

Return codes

The following return codes are possible:

0	Normal completion.
8	For keyed tables, the row specified by the value in key variables does not exist; the CRP is set to TOP (zero). For nonkeyed tables, the CRP was at TOP (zero) and remains at TOP.
12	Table is not open.
20	Severe error.

Example

Delete a row of the table TELBOOK.

```
"DGIEXEC TBDELETE TELBOOK"
```

or

```
CALL DGILINK ('TBDELETE','TELBOOK ');
```

TBEND — Close a table without saving

The TBEND service deletes the virtual storage copy of the specified table, making it unavailable for further processing. The permanent copy, if any, is not changed.

A TBEND request for a shared table causes the use count in the table for that logical screen to be decremented by 1. If the use count is zero, the TBEND service is performed. Otherwise, no action occurs, and the table is available for continued processing in any screen that still has a use count greater than zero.

REXX command format

DGIEXEC TBEND *table-name*

PL/I call format

CALL DGILINK ('TBEND   ', *table-name***);**

table-name
Specifies the name of the table to be ended.

Return codes

The following return codes are possible:

0	Normal completion
12	Table is not open
20	Severe error

Example

Delete the virtual storage copy table TELBOOK; do not change any permanent copy in the table library.

```
"DGIEXEC TBEND TELBOOK"
```

or

```
CALL DGILINK ('TBEND   ','TELBOOK ');
```

TBERASE — Erase a table

The TBERASE service deletes a table from the table library. The table library must be specified with this service.

The table must not be open in write mode when this service is invoked.

REXX command format

DGIEXEC TBERASE *table-name* [**LIBRARY(***lib***.***sublib***)**]
[**TYPE(***table-type***)**]

PL/I call format

CALL DGILINK ('TBERASE ', *table-name*[, *lib***.***sublib*
[, *table-type*]]**);**

table-name
Specifies the name of the table to be erased.

lib.sublib
Specifies the library that is to contain the table.

table-type
Specifies the type of the member in the library that is to contain the table.

Return codes

The following return codes are possible:

0	Normal completion
8	Table does not exist in the output library
12	Table in use
16	Table library could not be opened
20	Severe error

Example

Erase the table TELBOOK from the table library.

```
"DGIEXEC TBERASE TELBOOK"
```

or

```
CALL DGILINK ('TBERASE ','TELBOOK ');
```

TBEXIST — Determine whether a row exists in a table

The TBEXIST service tests for the existence of a specific row in a table with keys.

The current contents of the key variables, which are dialog variables that correspond to keys in the table, are used to search the table for the row.

This service is valid only for keyed tables. It causes the current row pointer (CRP) to be set to TOP (zero).

REXX command format

DGIEXEC TBEXIST *table-name*

PL/I call format

CALL DGILINK ('TBEXIST ', *table-name***);**

table-name
Specifies the name of the table to be searched.

Return codes

The following return codes are possible:

0	Normal completion; the CRP is positioned to the specified row.
8	For keyed tables, the specified row does not exist; the CRP is set to TOP (zero). For nonkeyed tables, this service is not possible; the CRP is set to TOP.
12	Table is not open.
20	Severe error.

Example

In the keyed table TELBOOK, test for the existence of a row having a specific key value.

```
"DGIEXEC TBEXIST TELBOOK"
```

or

```
CALL DGILINK ('TBEXIST ','TELBOOK ');
```

If the return code is 0, the row exists.

TBGET — Retrieve a row from a table

The TBGET service accesses a row in a table. If the NOREAD parameter is not specified, the row values are read into the function pool.

For tables with keys, the table is searched for the row to be fetched. The current contents of the key variables, which are dialog variables that correspond to keys in the table, are used as the search argument.

For tables without keys, the row pointed to by the current row pointer (CRP) is fetched. You can use the TBSCAN, TBSKIP, TBBOTTOM, and TBTOP services to position the CRP.

The CRP is always set to point to the row that was fetched.

All variables in the row, including any key and name variables, are stored in the corresponding dialog variables. A list of extension variable names can also be retrieved.

REXX command format

```
DGIEXEC TBGET table-name [SAVENAME(var-name)]
                         [ROWID(rowid-name)]
                         [NOREAD | SHARED]
                         [POSITION(crp-name)]
```

PL/I call format

```
CALL DGILINK ('TBGET   ', table-name[, var-name
                                    [, rowid-name
                                    [, 'NOREAD  ' | 'SHARED  '
                                    [, crp-name]]]]);
```

table-name
: Specifies the name of the table to be read.

var-name
: Specifies the name of a variable in which a list of extension variable names contained in the row is to be stored. The list is enclosed in parentheses, and the names within the list are separated by a blank.

rowid-name
: Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. This identifier can be specified later in the ROW parameter of TBSKIP to cause the CRP to be positioned to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

NOREAD
: Specifies that the variables contained in the requested row are not to be read into the variable pool.

SHARED
Specifies that the variables are to be read into the shared pool.

crp-name
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is set to TOP (zero), the row number returned is zero.

Return codes

The following return codes are possible:

0	Normal completion.
8	For keyed tables, the row specified by the value in the key variables does not exist; the CRP is set to TOP (zero). For nonkeyed tables, the CRP was set to TOP and remains at TOP.
12	Table is not open.
16	Variable value has been truncated, or insufficient space was provided to return all extension variable names.
20	Severe error.

Example

In the keyed table TELBOOK, from a row having a specific key value, copy variable values to the function pool variables having the same names.

```
"DGIEXEC TBGET TELBOOK"
```

or

```
CALL DGILINK ('TBGET   ','TELBOOK ');
```

TBMOD — Modify a row in a table

The TBMOD service unconditionally updates a row in a table.

For tables with keys, the table is searched for the row to be updated. The current contents of the key variables, which are dialog variables that correspond to keys in the table, are used as the search argument. If a match is found, the row is updated. If no match is found, TBADD adds a row at either of these places:

- The end of the table
- A point appropriate to maintaining the table in the sequence specified in a previously processed TBSORT request

For tables without keys, TBMOD is equivalent to TBADD. Any new row is added in either of these places:

- Immediately following the current row, pointed to by the current row pointer (CRP)
- At a point appropriate for maintaining the table in the sequence specified in a previously processed TBSORT request

The CRP is always set to point to the row that was updated or added.

The current contents of all *key* and *name* dialog variables that correspond to columns in the table are saved in the row.

Additional variables that were not specified when the table was created can also be saved in the row. These extension variables apply only to this row, not to the entire table. Whenever the row is updated, the extension variables must be respecified if they are to be rewritten.

REXX command format

```
DGIEXEC TBMOD table-name [SAVE(name-list)]
                         [ORDER]
```

PL/I call format

```
CALL DGILINK ('TBMOD   ', table-name[, name-list
                                    [, 'ORDER   ']]);
```

table-name
  Specifies the name of the table to be updated.

name-list
  Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created.

ORDER

Specifies that any new row is to be added or inserted in the order specified in the sort information record. A TBSORT must have been performed for this table prior to use of this keyword. For tables with keys, the row is updated and then reordered if necessary. If a match is not found or if the table does not have keys, the row is added at a point appropriate for maintaining the table in the sequence specified by the sort information record. This keyword is ignored if the table has never been sorted. If this keyword is omitted, any existing sort information record is nullified.

When a newly inserted row has sort field names equal to the sort field names of an existing row, the insertion is made after the existing row.

Return codes

The following return codes are possible:

0	Normal completion. For keyed tables, the existing row was updated. For nonkeyed tables, a new row was added to the table.
8	Keys did not match; new row added to the table. Returned only for tables with keys.
12	Table is not open.
16	Numeric conversion error; see numeric restrictions for TBSORT. Returned only for sorted tables.
20	Severe error.

Example

Update or add a row of variables in the table TELBOOK, using values from variables in the function pool.

```
"DGIEXEC TBMOD TELBOOK"
```

or

```
CALL DGILINK ('TBMOD   ','TELBOOK ');
```

TBOPEN — Open a table

The TBOPEN service reads a permanent table from the table library into virtual storage, and opens it for processing. TBOPEN should not be issued for temporary tables.

REXX command format

```
DGIEXEC TBOPEN table-name [WRITE | NOWRITE]
                          [SHARE]
                          [LIBRARY(lib.sublib)]
                          [TYPE(table-type)]
```

PL/I Call Format

```
CALL DGILINK ('TBOPEN  ', table-name[, 'WRITE   ' | 'NOWRITE '
                                    [, lib.sublib
                                    [, 'SHARE   '
                                    [, table-type]]]]);
```

table-name
: Specifies the name of the table to be opened.

WRITE
: Specifies that the table is being accessed for update. The updated table can later be saved on disk by the TBSAVE or TBCLOSE service. This option is the default.

The WRITE/NOWRITE usage of a shared table must be consistent on all TBOPEN and TBCREATE requests. That is, all requests for a given shared table that result in concurrent use of that table must specify the same WRITE or NOWRITE attribute.

NOWRITE
: Specifies read-only access. Upon completion of processing, the virtual storage copy should be deleted by invoking the TBEND or TBCLOSE service.

lib.sublib
: Specifies the library that is to contain the table to be opened.

table-type
: Specifies the type of the member in the library that is to contain the table.

SHARE
: Specifies that the table in virtual storage can be shared between logical screens while the display is in split-screen mode. The TBOPEN request from the first logical screen reads the table into virtual storage and opens it. Subsequent TBOPEN requests from other logical screens use the same table (and same CRP) that is in virtual storage.

A successful TBOPEN or TBCREATE request causes the use count in the table to be incremented by one. The use count determines the action taken by subsequent TBEND and TBCLOSE requests.

Return codes

The following return codes are possible:

0	Normal completion
8	Table does not exist
12	Table in use
16	Table library could not be opened
20	Severe error

Example

Access (open) the table TELBOOK for updating.

```
"DGIEXEC TBOPEN TELBOOK WRITE"
```

or

```
CALL DGILINK ('TBOPEN  ','TELBOOK ','WRITE   ');
```

TBPUT — Update a row in a table

The TBPUT service updates the current row of a table.

For tables with keys, the current contents of the key variables, which are dialog variables that correspond to keys in the table, must match the key of the current row, which is pointed to by the current row pointer (CRP). Otherwise, the update is not performed.

For tables without keys, the row pointed to by the CRP is always updated.

If the update was successful, the CRP remains unchanged. It continues to point to the row that was updated. The current contents of all dialog variables that correspond to columns in the table are saved in the row.

Additional variables that were not specified when the table was created can also be saved in the row. These extension variables apply only to the row, not to the entire table. Whenever the row is updated, the extension variables must be respecified if they are to be rewritten.

REXX command format

```
DGIEXEC TBPUT table-name [SAVE(name-list)]
                         [ORDER]
```

PL/I call format

```
CALL DGILINK ('TBPUT   ', table-name [, name-list
                                      [, 'ORDER   ']]);
```

table-name
: Specifies the name of the table to be updated.

name-list
: Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created.

ORDER
: Specifies that, if necessary, the updated row is to be moved to a point in the table that preserves the order specified in the sort information record. A TBSORT must have been performed for this table prior to use of this keyword. This keyword is ignored if the table has never been sorted. If this keyword is omitted, any existing sort information record is nullified.

  When a newly repositioned row has sort field-names equal to the sort field-names of an existing row, the row is inserted after the existing row.

Return codes

The following return codes are possible:

0	Normal completion.
8	For keyed tables, the key does not match that of the current row; the CRP is set to TOP (zero). For nonkeyed tables, the CRP was at TOP and remains at TOP.
12	Table is not open.
16	For sorted tables: numeric conversion error; see the discussion of numeric restrictions for TBSORT on page 197.
20	Severe error.

Example

Update a row in the table TELBOOK, using values from variables in the function pool.

```
"DGIEXEC TBPUT TELBOOK"
```

or

```
CALL DGILINK ('TBPUT   ','TELBOOK ');
```

TBQUERY — Obtain table information

The TBQUERY service returns information about a specified table, which must have been opened (TBOPEN) by the current user prior to invoking this service. The number and names of key fields, and the number and names of all other columns, can be obtained. The number of rows and the current row position can also be obtained.

All the parameters except *table-name* are optional. If all the optional parameters are omitted, TBQUERY simply validates the existence of an open table.

REXX command format

```
DGIEXEC TBQUERY table-name [KEYS(key-name)]
                           [NAMES(var-name)]
                           [ROWNUM(rownum-name)]
                           [KEYNUM(keynum-name)]
                           [NAMENUM(namenum-name)]
                           [POSITION(crp-name)]
                           [DESCRIPTION(description-name)]
```

PL/I call format

```
CALL DGILINK ('TBQUERY ', table-name[, key-name
                                    [, var-name
                                    [, rownum-name
                                    [, keynum-name
                                    [, namenum-name
                                    [, crp-name
                                    [, description-name]]]]]]]);
```

table-name
: Specifies the name of the table for which information is required.

key-name
: Specifies the name of a variable in which a list of key variable names contained in the table is to be stored. SDF2/DM encloses in parentheses a list that is not null. It separates the names within the list by a blank. If no key variables are defined for the table, the key-name variable is set to null.

var-name
: Specifies the name of a variable in which a list of variable names in the table is to be stored. This variable name was specified with the NAMES keyword when the table was created. SDF2/DM encloses the list in parentheses. It separates by a blank the names within a list that is not null. If no name variables are defined for the table, the *var-name* variable is set to null.

rownum-name
: Specifies the name of a variable in which the number of rows contained in the table is to be stored.

keynum-name
: Specifies the name of a variable in which the number of key variables contained in the table is to be stored.

namenum-name
: Specifies the name of a variable in which the number of variables in the table is to be stored. This variable name was specified with the NAMES keyword when the table was created.

crp-name
: Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is set to TOP (zero), the row number returned is zero.

description-name
: Specifies the name of a variable in which the description of the table is to be stored.

Return codes

The following return codes are possible:

0	Normal completion.
12	Table is not open.
16	Not all keys or names were returned, because insufficient space was provided.
20	Severe error.

Example

For the keyed table TELBOOK:

- In dialog variable QKEYS, store the names of key variables.
- In dialog variable QNAMES, store the names of nonkey variables.
- In dialog variable QROWS, store the number of rows.

```
"DGIEXEC TBQUERY TELBOOK KEYS(QKEYS) NAMES(QNAMES) ROWNUM(QROWS)"
```

or

```
CALL DGILINK ('TBQUERY ','TELBOOK ',
              'QKEYS ','QNAMES ','QROWS ');
```

TBSARG — Define a search argument

The TBSARG service establishes a search argument for scanning a table by using the TBSCAN service.

The direction of the scan, forward or backward, can be specified. The conditions that terminate the subsequent scan can also be specified.

The search argument is specified in dialog variables that correspond to columns in the table, including key variables. A null value ("") for one of the dialog variables means that the corresponding table variable is not to be examined during the search. However, the variable will be examined if the NOBSCAN parameter was specified when the variable was defined using the VDEFINE service.

TBSARG is normally used prior to TBSCAN operations to establish search arguments. To set up a search argument:

1. Set table variables in the function pool to nulls by using TBVCLEAR.
2. Set a value in each variable in the function pool that is to be part of the search argument.
3. Issue TBSARG to establish these variables as the search argument to be used in subsequently requested TBSCAN operations.

Use the NAMECOND list to establish search argument conditions. For any table variable that was given a value in the function pool but is not specified in the NAMECOND list, the default is EQ.

Only extension variables can be included in the search argument. They are included by specifying their names in the *name-list* parameter. The values of these variables become part of the search argument. A null value in an extension variable is a valid search argument.

A search argument of the form AAA* means that only the characters up to the * are compared. This is called a generic search argument. A generic search argument is specified by placing an asterisk in the last nonblank position of the argument. Asterisks embedded in the argument are treated as data. For example, to perform a generic search for a row value of DATA*12, the generic search argument is:

```
DATA*12*
```

The first asterisk is part of the search argument. The second asterisk designates the argument to be a generic search argument.

In a REXX EXEC, the following technique can be used to set a variable to a literal value that ends with an asterisk:

```
X = 'AAA*'
```

You can use either a DBCS or a MIX (DBCS and EBCDIC) format string as a search argument. If either is used as a generic search argument, it must be specified as follows:

- DBCS format string:

  *DBDBDBDB***

  where:

DBDBDBDB	Represents a 4-character DBCS string
**	Is a single DBCS character representing an asterisk (*)

- MIX (DBCS and EBCDIC) format string

  eeee[*DBDBDBDBDB*]*

  where:

eeee	Represents a 4-character EBCDIC string
DBDBDBDBDB	Represents a 5-character DBCS string
[	Represents a shift-out character
]	Represents a shift-in character
*	Is an asterisk in single-byte format

The position of the current row pointer (CRP) is not affected by the TBSARG service.

TBSARG replaces all previously set search arguments for the specified table.

For comparisons between the row values and the argument list, the values are considered character data, even if they represent numbers.

REXX command format

```
DGIEXEC TBSARG table-name [ARGLIST(name-list)]
                          [NEXT | PREVIOUS]
                          [NAMECOND(name-cond-pairs)]
```

PL/I call format

```
CALL DGILINK ('TBSARG  ', table-name[, name-list
                                    [, 'NEXT    ' | 'PREVIOUS'
                                    [, name-cond-pairs]]]);
```

table-name
: Specifies the name of the table for which an argument is to be established.

name-list
: Specifies a list of extension variables, by name, whose values are to be used as part of the search argument. See the discussion of name lists under “Parameter types for the DGILINK call” on page 6 for the specification of name lists.

NEXT
Specifies that the scan is to proceed from the row following the current row to the bottom of the table. This is the default.

PREVIOUS
Specifies that the scan is to proceed from the row preceding the current row to the top of the table. To scan the bottom row, the CRP must be positioned to TOP (zero).

name-cond-pairs
Specifies a list of names and conditions for determining the search argument conditions for scanning a table. One condition must be specified for each name in the list. This list is used to associate a particular operator (condition) with a previously established scan argument. This parameter does not affect how the search arguments are established.

The *name-cond-pairs* syntax is as follows:

Syntax

(*name***,***condition*[**,***name***,***condition*]...**)**

Each *name* must be the name of a key field, name field, or extension variable for the table. If the specified name does not exist, a severe error is reported.

The *condition* specifies the scan condition for the *name* (column) to which it is paired. The search arguments are specified in dialog variables that correspond to columns in the table; this determines the columns that take part in the search.

Valid conditions are EQ, NE, LE, LT, GE, and GT. If some or all condition-value pairs are not specified, the default EQ is used for those columns participating in the search. Each argument and its associated operator are treated as separate entities, not as subfields of a single argument. The following meanings are associated with the conditions:

EQ	Specifies that the search is for an equal condition between the argument value and the row value. This is the default.
NE	Specifies that the search is for a row value not equal to the argument value.
LE	Specifies that the search is for a row value less than or equal to the argument value.
LT	Specifies that the search is for a row value less than the argument value.
GE	Specifies that the search is for a row value greater than or equal to the argument value.
GT	Specifies that the search is for a row value greater than the argument value.

Return codes

The following return codes are possible:

0	Normal completion.
8	All column variables are null, and the *name-list* parameter was not specified; no argument is established.
12	Table is not open.
20	Severe error.

Example

Establish a search argument to be used by a TBSCAN operation of the table TELBOOK. Assume that LNAME and ZIPCODE are columns in table TELBOOK. Specify a scan direction of forward. Terminate the scan when the row value for the LNAME column is equal to JOHNSON and the ZIPCODE column is greater than 08007.

1. Invoke TBVCLEAR for table TELBOOK
2. Set variable LNAME to JOHNSON
3. Set variable ZIPCODE to 08007
4. Issue the following request:

```
"DGIEXEC TBSARG TELBOOK NEXT NAMECOND(LNAME,EQ,ZIPCODE,GT)"
```

or

```
CALL DGILINK ('TBSARG  ','TELBOOK ',' ','NEXT    ',
              '(LNAME,EQ,ZIPCODE,GT)');
```

TBSAVE — Save a table

The TBSAVE service writes the specified table from virtual storage to the table library. The table must be open in write mode.

When you store a table in a table library, you can give it a new name.

REXX command format

```
DGIEXEC TBSAVE table-name [NAME(alt-name)]
                          [LIBRARY(lib.sublib)]
                          [TYPE(table-type)]
```

PL/I call format

```
CALL DGILINK ('TBSAVE  ', table-name[, any
                                    [, alt-name
                                    [, any
                                    [, lib.sublib
                                    [, table-type]]]]]);
```

table-name
: Specifies the name of the table to be saved.

any
: Reserved.

alt-name
: Specifies an alternative name for the table. The table will be stored with the alternative name in the library. If another table with that name already exists in the table library, it is to be replaced. If the table being saved exists in the table library with the original name, that copy will remain unchanged.

lib.sublib
: Specifies the library that is to contain the table.

table-type
: Specifies the type of the member in the library that is to contain the table.

Return codes

The following return codes are possible:

0	Normal completion
12	Table is not open
16	Table library could not be opened
20	Severe error

Example

Write a table, TELBOOK, which was previously opened and currently in virtual storage, to the table library. Retain the copy in virtual storage for further processing. Do not close the table.

```
"DGIEXEC TBSAVE TELBOOK"
```

or

```
CALL DGILINK ('TBSAVE  ','TELBOOK ');
```

TBSCAN — Search a table

The TBSCAN service searches a table for a row with values that match an argument list. The argument list can be established by use of the TBSARG service, or specified in the *name-list* parameter for TBSCAN.

The search can be in either a forward or a backward direction. The forward direction starts with the row after the current row and continues to the end of the table. The backward direction starts with the row before the current row pointer (CRP) and continues to the top of the table. If a match is found, the CRP is set to that row. The row is retrieved unless the NOREAD parameter is specified. All variables in the row, including any keys and extension variables, are stored in the corresponding variables in the function pool. A list of extension variable names can also be retrieved.

Use of the *name-list* parameter is optional. If specified, it overrides the search argument set by the TBSARG service for this search only. The values of all variables specified in the *name-list* parameter become part of the search argument. Key, name, and extension variables can be specified.

A value of the form AAA* means that only the characters up to the * are compared. This is called a generic search argument. A generic search argument is specified by placing an asterisk in the last nonblank position of the argument. Asterisks embedded in the argument are treated as data. For example, to perform a generic search for a row value of DATA*12, the generic search argument is:

```
DATA*12*
```

The first asterisk is part of the search argument. The second asterisk designates the argument as a generic search argument. In a REXX EXEC, the following technique can be used to set a variable to a literal value that ends with an asterisk:

```
X = 'AAA*'
```

A null value in a variable is a valid search argument.

You can use either a DBCS or a MIX (DBCS and EBCDIC combined) format string as a search argument. If either is used as a generic search argument, it must be specified as follows:

- DBCS format string

  *DBDBDBDB***

  where:

DBDBDBDB	Represents a 4-character DBCS string
**	Is a single DBCS character representing the asterisk (*)

- MIX (DBCS and EBCDIC combined) format string

  eeee[*DBDBDBDBDB*]*

  where:

eeee	Represents a 4-character EBCDIC string
DBDBDBDBDB	Represents a 5-character DBCS string
[and]	Represent shift-out and shift-in characters
*	Is an asterisk in single-byte format

For comparisons between the row values and the argument list, the values are considered character data, even if they represent numbers.

REXX command format

```
DGIEXEC TBSCAN table-name [ARGLIST(name-list)]
                          [SAVENAME(var-name)]
                          [ROWID(rowid-name)]
                          [NEXT | PREVIOUS]
                          [NOREAD | SHARED]
                          [POSITION(crp-name)]
                          [CONDLIST(condition-value-list)]
```

PL/I call format

```
CALL DGILINK ('TBSCAN  ', table-name[, name-list
                                    [, var-name
                                    [, rowid-name
                                    [, 'NEXT    ' | 'PREVIOUS'
                                    [, 'NOREAD  ' | 'SHARED  '
                                    [, crp-name
                                    [, condition-value-list]]]]]]]);
```

table-name
: Specifies the name of the table to be searched.

name-list
: Specifies a list of key, name, or extension variables, by name, whose values are to be used as the search argument. Use of the *name-list* parameter is optional. If specified, it overrides the search argument set by the TBSARG service for this search only. If the *name-list* parameter is omitted, a search argument must have been established by a previous TBSARG command. Otherwise, a severe error occurs. See the discussion of name lists under “Parameter types for the DGILINK call” on page 6 for the specification of name lists.

Syntax

```
name[[,name]...]
```

var-name
Specifies the name of a variable in which a list of extension variable names contained in the row is to be stored. The list will be enclosed in parentheses, and the names within the list will be separated by a blank.

rowid-name
Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. Later, this identifier can be specified in the ROW parameter of TBSKIP to cause the CRP to be positioned to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

NEXT
Specifies that the scan is to proceed from the row following the current row to the bottom of the table. This is the default.

PREVIOUS
Specifies that the scan is to proceed from the row preceding the current row to the top of the table. To scan the bottom row, the current row pointer (CRP) must be positioned to TOP (zero).

NOREAD
Specifies that the variables contained in the requested row not be read into the variable pool.

SHARED
Specifies that the variables contained in the requested row be read into the shared pool.

crp-name
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returns to zero.

condition-value-list
Specifies a list of values for determining when the scan is to end. Each condition value relates to a search argument for a column or extension variable in the table as specified in the ARGLIST parameter. This parameter is ignored if no ARGLIST parameter is specified. The operators specified in the *condition-value-list* correspond one-to-one with the names in the ARGLIST. If there are extra operators, a severe error condition is reported.

Syntax

condition[[**,** *condition*]...]

The valid conditions are EQ, NE, LE, LT, GE, and GT. If there are fewer conditions than search arguments, the default EQ is used for those columns. Each argument and its associated operator are treated as separate entities, and not as subfields of a single argument.

The following meanings are associated with the condition values:

EQ	Specifies that the scan is to end when an equal condition exists between the argument value and the row value. This is the default.
NE	Specifies that the scan is to end when the row value is not equal to the argument value.
LE	Specifies that the scan is to end when the row value is less than or equal to the argument value.
LT	Specifies that the scan is to end when the row value is less than the argument value.
GE	Specifies that the scan is to end when the row value is greater than or equal to the argument value.
GT	Specifies that the scan is to end when the row value is greater than the argument value.

Return codes

The following return codes are possible:

0	Normal completion.
8	Row does not exist, no match was found; the CRP is set to TOP (zero). The row ID remains unchanged.
12	Table is not open.
16	Variable value has been truncated, or insufficient space is provided to return all extension variable names.
20	Severe error.

Example 1

For the table TELBOOK, move the CRP for table TELBOOK to the row that fulfills the search argument specified in a preceding TBSARG operation. For an example of TBSARG, see “Example” on page 186. Copy values from variables in that row to function pool variables whose names match those of the table variables.

```
"DGIEXEC TBSCAN TELBOOK"
```

or

```
CALL DGILINK ('TBSCAN  ','TELBOOK ');
```

Example 2

For the table TELBOOK, use the TBSCAN service to position the CRP of table TELBOOK to the row containing the name JOHNSON in variable LNAME, and the zip code 08007 in variable ZIPCODE. Copy values of the variables in that row to function pool variables whose names match those of the table variables.

1. Set function pool variable LNAME to JOHNSON.
2. Set function pool variable ZIPCODE to 08007.
3. Issue the following request:

   ```
   "DGIEXEC TBSCAN TELBOOK ARGLIST(LNAME,ZIPCODE)"
   ```

   or

   ```
   CALL DGILINK ('TBSCAN  ','TELBOOK ','(LNAME,ZIPCODE)');
   ```

If the return code is 0, the row was found and values were copied from the row variables to function pool variables.

TBSKIP — Move the row pointer

The TBSKIP service moves the current row pointer (CRP) of a table forward or backward by a specified number of rows and retrieves the row to which it is pointing, unless the NOREAD parameter is specified.

All variables in the row, including any keys and extension variables, are stored in the corresponding dialog variables. A list of extension variable names can also be retrieved.

REXX command format

```
DGIEXEC TBSKIP table-name [NUMBER(number)]
                          [SAVENAME(var-name)]
                          [ROWID(rowid-name)]
                          [ROW(rowid)]
                          [NOREAD | SHARED]
                          [POSITION(crp-name)]
```

PL/I call format

```
CALL DGILINK ('TBSKIP  ', table-name[, number
                                    [, var-name
                                    [, rowid-name
                                    [, rowid
                                    [, 'NOREAD  ' | 'SHARED  '
                                    [, crp-name]]]]]]);
```

table-name
: Specifies the name of the table to be used.

number
: Specifies the direction and number of rows to move the CRP. This parameter must be a positive or negative integer. A positive integer moves the CRP toward the bottom of the table. A negative integer moves it toward the top. Zero is a valid value: it retrieves the current row.

  For a call, this parameter must be a fullword fixed binary number.

  A default skip of +1 is used if both the ROW and NUMBER parameters are omitted. When the ROW parameter is specified, no default skip of +1 is assumed if the NUMBER parameter is omitted.

var-name
: Specifies the name of the variable in which a list of extension variable names contained in the row is to be stored. The list is enclosed in parentheses; the names within the list are separated by a blank.

rowid-name
Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. This identifier can be specified later in the ROW parameter to cause the CRP to be positioned to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

rowid
Specifies the numeric value that uniquely identifies the row to be accessed. This value is obtained by using the *rowid-name* parameter.

A default skip of +1 is used if both the ROW and NUMBER parameters are omitted. When the ROW parameter is specified, no default skip of +1 is assumed if the NUMBER parameter is omitted.

NOREAD
Specifies that the variables contained in the requested row not be read into the variable pool.

SHARED
Specifies that the variables contained in the requested row be read into the shared pool.

crp-name
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returned is zero.

Return codes

The following return codes are possible:

0	Normal completion.
8	CRP would have gone beyond the number of rows in the table. This includes a table empty condition, with CRP set to TOP (zero). The row ID remains unchanged.
12	Table is not open.
16	Variable value has been truncated, or insufficient space is provided to return all extension variable names.
20	Severe error.

Example

In the table TELBOOK, move the current row pointer (CRP) to the next row. Then, copy values from variables in that row to variables in the function pool whose names match those of the variables in the row.

```
"DGIEXEC TBSKIP TELBOOK"
```

or

```
CALL DGILINK ('TBSKIP  ','TELBOOK ');
```

TBSORT — Sort a table

The TBSORT service places the rows of an open table in a user-specified order and stores this specified order in a sort information record.

The sort can be done on more than one field and in either an ascending or a descending order. TBSORT can be issued for an empty table. When a TBSORT is completed, the CRP is set to zero (TOP).

The sort information record is maintained with the table. This record contains the order of the last sort and provides for rows to be added to the table in the proper sequence after a sort has been completed. This is done through the ORDER keyword on the TBADD, TBMOD, and TBPUT services. The sort information record is saved on external storage when a TBSAVE or TBCLOSE operation successfully completes. It is retrieved during TBOPEN processing.

REXX command format

DGIEXEC TBSORT *table-name* **FIELDS(***sort-list***)**

PL/I call format

CALL DGILINK ('TBSORT ', *table-name*, *sort-list***);**

table-name
Specifies the name of the table to be sorted.

sort-list
Specifies sort fields.

Syntax

field-name, {**B** | **C** | **N**}, {**A** | **D**}, [[*field-name*, {**B** | **C** | **N**}, {**A** | **D**},]...]

Each sort *field-name* must be either a KEY field or a NAME field. The first (leftmost) *field-name* is the primary key (most significant). The rows are collated in accordance with the values of the *field-names* parameters.

The *field-name* parameter is followed by a sort field type designator. The sort field type designator can have a value of C for a character sort, N for a numeric sort, or B for a binary sort. For English, where sorting is in EBCDIC sequence, specifying either C or B as the sort field type designator causes the same sort order. For other languages, in which the character format can be other than EBCDIC, specify B for a binary sort.

The collating sequence for character sorts during DBCS and English sessions is in EBCDIC order. This means, for example, that all lowercase letters precede uppercase letters when sorting in an ascending sequence. For other languages, characters are sorted such that both uppercase and lowercase, as well as accented and nonaccented versions of a letter, are sorted in the proper order.

The sort field type designator is followed by a sort sequence direction value. The sort sequence direction value can be either A (ascending) or D (descending). The field type designator and the sort sequence direction can be omitted for only the last named field. The field type designator defaults to C (character); the sort sequence direction defaults to A (ascending).

In some countries, the comma is used as a decimal point. To accommodate various national usages, three numeric representations are supported: period, comma, and French. These are shown in Figure 52.

The TBSORT service accommodates these numeric representations. The convention used is determined by the language of the session, specified by the value of ZLANG in the system profile table. The current English version accepts only the period, treating it as the delimiter of the whole and decimal portion of a number. Sorting is based on the specified language convention.

The following restrictions apply to fields for a numeric type sort:

1. The field must be a decimal number and optionally can contain a plus (+) or minus (−) sign. The decimal number can be either a whole number, such as 234, or a mixed number, such as 234.56, composed of a whole number followed by a decimal point. A decimal point is not required after a whole number, but is required in a mixed number. (Under the period convention, the decimal point is represented by a period (.); under the comma or French conventions, the decimal point is represented by a comma (,).) No other characters are allowed, except leading blanks.
2. The string can have leading blanks following the sign character.
3. No numeric string can exceed 16 characters. This length value includes any plus or minus sign, any blanks, or any decimal point.
4. The largest value that can be sorted is plus or minus 2147483647.

Figure 52. Decimal point representations

Convention	Example	Example	Where used
Period	1,234.56	0.789	Japan, Mexico, U.K., U.S.A.
Comma	1.234,56	0,789	Most other countries
French	1234,56	0,789	France, South Africa

Return codes

The following return codes are possible:

0	Normal completion
12	Table is not open
16	Numeric convert error
20	Severe error

Notes on performance:

1. The poorest performance for this sorting algorithm occurs when sorting a table that is already in order. This situation can be avoided by sorting a table when it is empty and then adding rows in order, thus avoiding the need to sort the table, or by first sorting the table on another key to randomize the table before sorting on the desired fields.
2. A numeric sort affects performance because two numbers must be converted for each comparison.

Example 1

Sort the LASTNAME field for table TELBOOK. Use the defaults of A (ascending) and C (character).

```
"DGIEXEC TBSORT TELBOOK FIELDS(LASTNAME)"
```

or

```
CALL DGILINK ('TBSORT  ','TELBOOK ','LASTNAME');
```

Example 2

Perform a sort on table MODSIZES. Sort the character field NAME in ascending sequence. Then sort the numeric field SIZE in descending sequence.

```
"DGIEXEC TBSORT MODSIZES FIELDS(NAME,C,A,SIZE,N,D)"
```

or

```
CALL DGILINK ('TBSORT  ','MODSIZES','(NAME,C,A,SIZE,N,D)');
```

TBTOP — Set the current row pointer to the top of a table

The TBTOP service sets the current row pointer (CRP) to the top of a table, ahead of the first row.

REXX command format

DGIEXEC TBTOP *table-name*

PL/I call format

CALL DGILINK ('TBTOP   ', *table-name***);**

table-name Specifies the name of the table to be used.

Return codes

The following return codes are possible:

0	Normal completion
12	Table is not open
20	Severe error

Example

For the table TELBOOK, move the current row pointer (CRP) to the row immediately before the first row.

```
"DGIEXEC TBTOP TELBOOK"
```

or

```
CALL DGILINK ('TBTOP   ','TELBOOK ');
```

TBVCLEAR — Clear table variables

The TBVCLEAR service sets dialog variables to nulls.

All dialog variables that correspond to columns in the table, which were specified when the table was created, are cleared.

The contents of the table and the position of the current row pointer (CRP) are not changed by this service.

REXX command format

DGIEXEC TBVCLEAR *table-name* [**SHARED**]

PL/I call format

CALL DGILINK ('TBVCLEAR', *table-name*)[, **'SHARED '**];

table-name	Specifies the name of the table to be used.
SHARED	Specifies that the variables are to be deleted from the shared pool.

Return codes

The following return codes are possible:

0	Normal completion
12	Table is not open
20	Severe error

Example

Clear dialog variables associated with the table TELBOOK.

```
"DGIEXEC TBVCLEAR TELBOOK"
```

or

```
CALL DGILINK ('TBVCLEAR','TELBOOK ');
```

VCOPY — Create a copy of a variable

The VCOPY service is not available in REXX application prototypes.

The VCOPY service copies the values of variables to user storage, or returns their location without copying them. A length array and a value array, defined previously in your application prototype, determine the locations in user storage where the values are to be put. After the VCOPY call, the length array contains the actual length of each value.

SDF2/DM looks for a value first in the function pool, then in the shared pool, then in the profile pool. It copies the first value it finds.

PL/I call format

```
CALL DGILINK ('VCOPY   ', name-list , length-array
                                    , value-array
                                    [, 'MOVE    ' | 'LOCATE  ']);
```

name-list
: Specifies the variable names to be copied to user storage. The variable names can be specified as a single 8-character value, a list enclosed in parentheses, or a *name-list* structure.

length-array
: Specifies the length array.

  On input, this array of fullword fields must contain the lengths of the data areas within the value array.

  On output, each element of the array is set to the number of bytes of data for the corresponding variable. The length does not include trailing blanks unless the variable is defined to maintain blanks, such as when you use VCOPY to copy a variable that was defined using VDEFINE with the NOBSCAN option.

value-array
: Specifies the value array.

  Values are written in this array of pointers. The location of each value is determined from the lengths specified in the length array.

MOVE
: Specifies that the variable is to be copied. This is the default.

LOCATE
: Specifies that the variable is not to be copied, but the address of the variable in the function pool or in the shared pool is to be returned. This is valid only for character-type variables.

Return codes

The following return codes are possible:

0	Normal completion.
8	One or more variables do not exist.
12	Validation failed.
16	Truncation occurred during data movement.
20	Severe error.

Example

Copy the values of NAME, PHONE, and ADDRESS to array VARR1. Array LARR1 contains the lengths reserved for each variable in VARR1.

```
DCL 1 VARR1,
     2 NAME CHAR(8),
     2 PHONE CHAR(10),
     2 ADDRESS CHAR(40);
DCL LARR1(3) FIXED BIN(31);
LARR1(1)=8;
LARR1(2)=10;
LARR1(3)=40;
CALL DGILINK ('VCOPY   ','(NAME PHONE ADDRESS)',LARR1,VARR1);
```

VDEFINE — Define function variables

The VDEFINE service is not available in REXX application prototypes.

The VDEFINE service creates a variable name without a variable value in the function pool. The variable name points instead to a variable in user storage that contains the actual value.

If the variable name already exists in the function pool, you create a new version of the variable. This new version temporarily overrides any other versions, until you erase it using the VDELETE service.

If several versions of the variable exist, VDELETE deletes only the most recent version. The previous version is then used. For example, in a subroutine you can use VDEFINE to create a variable named XXX and later use VDELETE to delete it. If XXX already exists, its previous value will be available again when you issue the VDELETE call at the end of the subroutine.

You can define a list of dialog variables with a single call to the VDEFINE service. The variables in user storage must be in contiguous locations or be defined as an array or structure within the program.

A variable that is created with VDEFINE can be translated into a bit string, fixed binary string, or hexadecimal string. The translation occurs automatically when the variable is stored by an SDF2/DM service. Translation back to character string format occurs automatically when the variable is accessed.

When a defined variable is stored, either of two errors can occur:

Truncation If the current length of the variable is greater than the defined length within the module, the remaining data is lost.

Translation If the variable is defined as something other than a character string, and if the external representation has invalid characters, the contents of the defined variable are lost.

In either case, the SDF2/DM service issues a return code of 16.

PL/I call format

```
CALL DGILINK ('VDEFINE ', name-list , variable-arr-name
                                    , format
                                    , length
                                    [[, dimension]
                                    [, 'NOBSCAN '|'LIST    '|'COPY    ']]);
```

name-list
Specifies the variable name or variable names (set of variables) to be created in the function pool. If the LIST parameter is specified, each variable of this set can have a different format and length. If the dimension parameter is specified each variable is considered as an array variable. Each variable name in the function pool will point to a variable in user storage.

variable-arr-name
Specifies the name of a variable in user storage or the name of an array that contains a list of variable names. A variable in user storage is pointed to by a variable name in the function pool.

format
Specifies a data conversion format that applies to all variables or, if you specify the LIST keyword, the name of an array that contains a list of data conversion formats.

Valid formats are:

BIT Bit string, represented by the characters 0 or 1. Within the variable, the data is left-justified and padded on the right with binary zeros. For these variables, a null value is stored as binary zeros and cannot be distinguished from a zero value.

CHAR Character string. Within the variable, the data is left-justified and padded on the right with blanks.

No data conversion is performed when fetching and storing a CHAR variable, neither is there any checking for valid characters. In PL/I, a character string to be used as a dialog variable must be declared as fixed length, because VDEFINE cannot distinguish variable-length PL/I strings.

FIXED Fixed binary integer, represented by the characters 0 through 9.

Fixed variables that have a length of 4 bytes (fullword) are treated as signed, which is represented by the absence or presence of a leading minus sign (–). They can also have a null value, which is stored as the maximum negative number (X'80000000').

Fixed variables that have a length of less than 4 bytes are treated as unsigned. For these variables, a null value is stored as binary zeros, and cannot be distinguished from a zero value.

HEX Bit string, represented by the characters 0–9 and A–F. Within the variable, the data is left-justified and padded on the right with binary zeros. For these variables, a null value is stored as binary zeros and cannot be distinguished from a zero value.

length
Specifies a length value that applies to all variables or, if you specify the LIST keyword, the name of an array that contains a list of lengths. The length is specified in bytes and must be a fullword fixed binary integer. The maximum length of a fixed binary integer (FIXED) variable is 4 bytes. The maximum length for other types of variables is 32767 bytes.

For character variables in a C program, this length should be one less than the length of the program variable. This allows for the null terminator at the end of the string. Always initialize variables for the length specified in this parameter.

dimension
Defines the number of replications of the variable or set of variables specified in the *name-list*. This parameter must be a fullword fixed binary integer. The value must be in the range 1–65534.

NOBSCAN
Specifies that any trailing blanks in the variables are to remain in the variables.

LIST
Specifies that the variables in the *name-list* parameter structure have different formats (format array) and lengths (length array).

To define only selected variables, using VDEFINE:

- Specify the LIST option.
- Specify an asterisk (*) in the *name-list* parameter for variables to be skipped.
- Specify an asterisk in the corresponding position in the format-definition array for those portions of user storage that are to be ignored by VDEFINE.

The asterisk lets SDF2/DM determine the address of the user storage of the next true variable name in the *name-list*. This is determined by the corresponding length in the *length* parameter array.

COPY
Specifies that any dialog variable with the same name can be used to initialize the defined storage.

Return codes

The following return codes are possible:

0	Normal completion
8	Variable not found
16	Data truncation occurred
20	Severe error

Example 1

Create in the function pool a variable named MSGNAME that points to the value of the variable ERRMSG in user storage. The field is a character string 8 bytes long. Program variable L8 contains a value of 8.

```
CALL DGILINK ('VDEFINE ','(MSGNAME)',ERRMSG,'CHAR    ',L8);
```

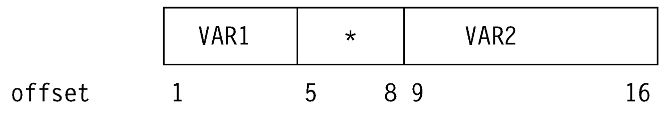
Example 2

Define three variables—FVAR, CVAR, and DVAR—with data formats of FIXED, CHAR, and HEX, and with lengths of 4, 5, and 20, respectively.

```
DECLARE

 1 VARS,
   3 FVAR     FIXED BIN(31),
   3 CVAR     CHAR(5),
   3 DVAR     CHAR(20),
 FARR(3) CHAR(8),
 LARR(3) FIXED BIN(31);

FARR(1) = 'FIXED';
FARR(2) = 'CHAR';
FARR(3) = 'HEX';
LARR(1) = 4;
LARR(2) = 5;
LARR(3) = 20;

CALL DGILINK ('VDEFINE ','(FVAR CVAR DVAR)',VARS,FARR,LARR,'LIST    ');
```

Example 3

Define two dialog variables, VAR1 and VAR2, contained in a structure named STRCVARS. The structure contains other data that is not used.



```
  DECLARE

    1 STRCVARS,
       3 VAR1 FIXED BIN(31),
       3 FILLER CHAR(4),
       3 VAR2 CHAR(8)
    FARR(3) CHAR(8),
    LARR(3) FIXED BIN(31);

FARR(1) = 'FIXED   ';
FARR(2) = '*';
FARR(3) = 'CHAR    ';
LARR(1) = 4;
LARR(2) = 4;
LARR(3) = 8;

CALL DGILINK('VDEFINE','(VAR1 * VAR2)',STRCVARS,FARR,LARR,'LIST     ');
```

Example 4

Define the dialog variables for a scrollable list, which is defined as follows on the panel:

```
)LIST name TOPROW(toprow) NUMROWS(rows) MODVAR(modlns)
$VAR1     $VAR2 $VAR3              +
```

```
DCL
  SIZE         FIXED   BIN(15) INIT(500), /* MAXIMUM ROWS       */
  1 ARRAY(SIZE),                          /* ARRAY FOR ROWS     */
   3 VAR1      FIXED BIN(31),             /* 1ST COLUMN         */
   3 VAR2      CHAR(5),                   /* 2ND COLUMN         */
   3 VAR3      CHAR(20),                  /* 3RD COLUMN         */
  TYPE(3)      CHAR(8),                   /* FIELD TYPE ARRAY   */
  LEN (3)      FIXED BIN(31);             /* FIELD LENGTH ARRAY */

DCL
  TOPROW       FIXED   BIN(31,0),         /* TOPROW(toprow)     */
  ROWS         FIXED   BIN(31,0),         /* NUMROWS(rows)      */
  MODLNS(SIZE) FIXED   BIN(31,0),         /* MODVAR(modlns)     */
  FBIN         CHAR(8) INIT('FIXED   '),  /* FIELD TYPE         */
  L4           FIXED   BIN(31,0) INIT(4); /* FIELD LENGTH       */

TYPE(1)  = 'FIXED';                       /* SETUP FIELD TYPES  */
TYPE(2)  = 'CHAR ';
TYPE(3)  = 'CHAR ';
LEN(1)   = 4;                             /* SETUP FIELD LENGTH */
LEN(2)   = 5;
LEN(3)   = 20;

CALL DGILINK('VDEFINE ','(VAR1 VAR2 VAR3)',array,type,len,size,'LIST');
CALL DGILINK('VDEFINE ','TOPROW ', toprow,fbin,l4);
CALL DGILINK('VDEFINE ','ROWS '  , rows,  fbin,l4);
CALL DGILINK('VDEFINE ','MODLNS ', modlns,fbin,l4,size);
```

VDELETE — Remove a definition of function variables

The VDELETE service is not available in REXX application prototypes.

The VDELETE service erases a variable that was created with the VDEFINE service.

PL/I call format

CALL DGILINK ('VDELETE ', *name-list* | **'*   ');**

name-list
: Specifies the variable names to be erased. The variable names can be specified as a single 8-character value, a list enclosed in parentheses, or a *name-list* structure. Specify an asterisk (*) to remove all the variables that were created using VDEFINE.

Return codes

The following return codes are possible:

0	Normal completion
8	At least one variable not found
20	Severe error

Example

Erase variable MSGNAME, which was created with VDEFINE.

```
CALL DGILINK ('VDELETE ','MSGNAME ');
```

VERASE — Remove variables from shared or Profile Pool

The VERASE service removes variable names and values from the shared pool or the profile pool, or both. Any value in the function pool (for languages other than REXX) is also erased. Pointers for variables that were defined with VDEFINE are not erased.

REXX command format

DGIEXEC VERASE *name-list* [**ASIS** | **SHARED** | **PROFILE** | **BOTH**]

PL/I call format

```
CALL DGILINK ('VERASE  ', name-list[, 'ASIS    ' |
                                      'SHARED  '|
                                      'PROFILE '|
                                      'BOTH    '] );
```

name-list
: Specifies the dialog variable names to be erased. The variable names can be specified as a single 8-character value, a list enclosed in parentheses, or a *name-list* structure.

ASIS
: Specifies that the variable is to be erased from the shared pool, if it exists there. If the variable does not exist in the shared pool, it is erased from the profile pool. This is the default.

SHARED
: Specifies that the variable is to be erased from the shared pool.

PROFILE
: Specifies that the variable is to be erased from the profile pool.

BOTH
: Specifies that the variable is to be erased from both the shared pool and the profile pool.

Return codes

The following return codes are possible:

0	Normal completion
8	At least one variable not found
20	Severe error

Note: SDF2/DM processes the entire name list, even if it cannot find one or more of the variable names in the list.

Example

Remove variables NAME, PHONE, and ADDRESS from the profile pool.

```
"DGIEXEC VERASE (NAME PHONE ADDRESS) PROFILE"
```

or

```
CALL DGILINK ('VERASE  ','(NAME PHONE ADDRESS)','PROFILE ');
```

VGET — Retrieve variables from a pool or profile

The VGET service gets the value of one or more variables from the shared pool, from the profile pool, or from the global pool. There are some differences between REXX application prototypes and non-REXX application prototypes:

- In a REXX application prototype, the value in the shared pool or the profile pool is copied to a REXX variable.
- In a non-REXX application prototype, the value in the shared pool or the profile pool is copied to the function pool. For a variable that was defined with VDEFINE, the value in user storage is updated. For a system variable, whose name starts with the letter Z, VGET with the PROFILE keyword copies a value to the shared pool instead of to the function pool.

  Note: User-defined variables whose names start with Z are handled as system variables.

REXX command format

DGIEXEC VGET *name-list* [**ASIS** | **SHARED** | **PROFILE** | **GLOBAL**]

PL/I call format

```
CALL DGILINK ('VGET    ', name-list[, 'ASIS    ' |
                                      'SHARED  ' |
                                      'PROFILE ' |
                                      'GLOBAL  '] );
```

name-list
: Specifies the variable names to be copied. The variable names can be specified as a single 8-character value, a list enclosed in parentheses, or a *name-list* structure.

ASIS
: Specifies that the variable is to be copied from the shared pool, if it exists there. If the variable does not exist in the shared pool, it is copied from the profile pool. This is the default.

SHARED
: Specifies that the variable is to be copied from the shared pool.

PROFILE
: Specifies that the variable is to be copied from the profile pool.

GLOBAL
: Specifies that the variable is to be copied from the global pool.

Return codes

The following return codes are possible:

0	Normal completion
8	Variable not found
16	Translation error or truncation occurred during data movement
20	Severe error

Note: If a variable that you are trying to copy does not exist, the variable to which you are copying is set to blanks or nulls.

Example

Copy the values of NAME, PHONE, and ADDRESS from the profile pool.

```
"DGIEXEC VGET (NAME PHONE ADDRESS) PROFILE"
```

or

```
CALL DGILINK ('VGET    ','(NAME PHONE ADDRESS)','PROFILE ');
```

VPUT — Update variables in the shared, profile, or global pool

The VPUT service copies the value of one or more variables to the shared pool, to the profile pool, or to the global pool. There are some differences between REXX application prototypes and non-REXX application prototypes:

- In a REXX application prototype, the value is copied from a REXX variable. If no REXX variable with the specified name exists, VPUT with the PROFILE keyword copies a value from the shared pool.
- In a non-REXX application prototype, the value is copied from the function pool. For a variable that was defined with VDEFINE, the value in user storage is copied. For a system variable, whose name starts with the letter Z, VPUT with the PROFILE keyword copies a value from the shared pool instead of from the function pool.

  Note: User-defined variables whose names start with Z are handled as system variables.

REXX command format

DGIEXEC VPUT *name-list* [**ASIS** | **SHARED** | **PROFILE** | **GLOBAL**]

PL/I call format

```
CALL DGILINK ('VPUT    ', name-list[, 'ASIS    ' |
                                      'SHARED  ' |
                                      'PROFILE ' |
                                      'GLOBAL  '] );
```

name-list
: Specifies the variable names to be copied. The variable names can be specified as a single 8-character value, a list enclosed in parentheses, or a *name-list* structure.

ASIS
: Specifies that the value in the shared pool (if any) is to be updated. If the variable does not exist in the shared pool, any value in the profile pool is updated. If the variable exists in neither the shared pool nor the profile pool, a new variable is created in the shared pool. This is the default.

SHARED
: Specifies that the variable is to be copied to the shared pool.

PROFILE
: Specifies that the variable is to be copied to the profile pool.

GLOBAL
: Specifies that the variable is to be copied to the global pool.

Return codes

The following return codes are possible:

0	Normal completion
8	Variable not found
16	Truncation occurred while copying variables
20	Severe error

Example

Copy the values of NAME, PHONE, and ADDRESS to the profile pool.

```
"DGIEXEC VPUT (NAME PHONE ADDRESS) PROFILE"
```

or

```
CALL DGILINK ('VPUT    ','(NAME PHONE ADDRESS)','PROFILE ');
```

VREPLACE — Replace a variable

The VREPLACE service is not available in REXX application prototypes.

The VREPLACE service copies the values of variables in user storage to the function pool. System variables, whose names begin with the letter Z, are copied to the shared pool instead of to the function pool. A length array and a value array, which were defined previously in your application prototype, determine the locations of the values in user storage.

PL/I call format

CALL DGILINK ('VREPLACE', *name-list* **,** *length-arr-name***,** *value-arr-name***);**

name-list
: Specifies the variable names to be copied from user storage. The variable names can be specified as a single 8-character value, a list enclosed in parentheses, or a *name-list* structure.

length-arr-name
: Specifies the name of a length array. On input, this array of fullword fields must contain the lengths of the data areas within the value array.

value-arr-name
: Specifies the name of the value array in user storage that contains the values. The location of each value is determined from the lengths specified in the length array.

Return codes

The following return codes are possible:

0	Normal completion.
8	One or more variables do not exist.
16	Truncation occurred during data movement.
20	Severe error.

Example

Copy the values of NAME, PHONE, and ADDRESS to the function pool from array VARR1. Array LARR1 contains the lengths reserved for each variable in VARR1.

```
DCL 1 VARR1,
     2 NAME CHAR(8),
     2 PHONE CHAR(10),
     2 ADDRESS CHAR(40);
DCL LARR1(3) FIXED BIN(31);
LARR1(1)=8;
LARR1(2)=10;
LARR1(3)=40;
CALL DGILINK ('VREPLACE','(NAME PHONE ADDRESS)',LARR1,VARR1);
```

VRESET — Reset function variables

The VRESET service is not available in REXX application prototypes.

The VRESET service erases the function pool. It also erases pointers to variables that were created using VDEFINE.

PL/I call format

```
CALL DGILINK ('VRESET  ');
```

Return codes

The following return codes are possible:

0	Normal completion
20	Severe error

Chapter 9. System commands

Within your dialog, a user can issue commands in any of the ways described under "Commands and function keys" on page 7.

The following system commands are available within SDF2/DM:

ACTIONS
Moves the cursor between the action bar and the panel body.

CANCEL
Cancels the current operation, as follows:

- From a pull-down, the CANCEL command removes the pull-down, then positions the cursor on the first action-bar choice.
- From a CUA panel that was defined with a)PANEL statement, the CANCEL command sets the system variable ZVERB to the string CANCEL, then ends the DISPLAY service with return code 8.
- From a panel with no)PANEL statement, the CANCEL command sets system variable ZCMD to the string CANCEL, then ends the DISPLAY service with return code 0.

CMDLINE [BOTTOM|ASIS]
This command applies only to panels whose command line is coded within the first six lines of the panel definition. On those panels, it moves the command line to the top or bottom of the panel:

BOTTOM	Moves the command line to the bottom of the panel. If the command line is followed by a text line, that line is assumed to be the message line and is moved with the command line.
ASIS	Leaves the command line in the position in which it appears within the panel definition.

The CMDLINE value is stored in the ZPLACE system variable.

CRETRIEV
The action of the CRETRIEV (conditional retrieve) command depends on the position of the cursor when you enter the command.

- If the cursor is within the command line, the CRETRIEV command does the same as the RETRIEVE command: it displays a previously issued command in the command field.
- If the cursor is not within the command field, the CRETRIEV command does the same as the CURSOR command: it moves the cursor to the command line or, if no command line is available, to the first input field.

CUAATTR
Displays a panel on which you can change the colors, intensities, and highlighting for CUA panel element attributes.

CURSOR
Moves the cursor to the command line or, if no command line is available, to the first input field.

DOWN
Scrolls toward the bottom of the data. For more information, see “Using the scrolling commands” on page 222.

END
Ends the current operation, as follows:

- From a help panel, it ends the panel display and redisplays the panel from which help was requested.
- From any other panel, it ends the DISPLAY service with return code 8.

EXHELP
Provides general information about the contents of a panel.

EXIT
Exits the current operation, as follows:

- From a help panel, terminates the panel display and redisplays the panel from which help was requested.
- From a CUA panel that was defined with a)PANEL statement, sets system variable ZVERB to the string EXIT, then ends the DISPLAY service with return code 8.
- From a panel with no)PANEL statement, sets system variable ZCMD to the string EXIT, then ends the DISPLAY service with return code 0.

FKA
Toggles the display of the function key area. The first time you enter the FKA command, the keys are removed from the display. If you enter the command again, the function key area is displayed.

HELP
Displays additional information about a field, message, or reference phrase.

KEYS
Displays a panel on which you can view and change function key definitions, and change the way text is displayed.

If there is no keylist, you can use the KEYS command to set the values of function keys 1 through 12. Any function key from 13 through 24 always has the same effect as the corresponding function key from 1 through 12.

If there is a keylist, the KEYS command sets the values of variables ZPF01 through ZPF12. (This has an effect only if you use these variables in your keylist.) In Figure 48 on page 110, for example, these variables control function keys 13 through 24.

With the KEYS command, you can set the terminal type to either 3270 or 3270KN. Use 3270KN for a terminal that supports Japanese Katakana characters. With 3270KN, all text is translated to uppercase. This value is stored in the ZTERM system variable.

You can also specify whether graphic characters are used for the action-bar separator, pull-down window frames, and pop-up window frames. This value is stored in the ZGBORDER system variable.

KEYSHELP
Provides a brief description of each key defined for a panel.

LEFT
Scrolls left. For more information, see “Using the scrolling commands” on page 222.

POPUP *panel-name*
The POPUP command displays a pop-up panel at the location .CURSOR. You can use it to prompt for options from the)PROC section without altering the application prototype, for example, as follows:

```
)PROC
  ...
  .CURSOR = OPTFLD
  .ZCMD = 'POPUP OPTGET'
```

PANELID [ON|OFF]
Specifies whether the panel name is to be displayed:

ON Displays the panel name (the name of the file that contains the panel definition) in the top left-hand corner of the display.

OFF Removes the panel name.

PANELID with no parameter toggles the panel name on and off.

If an action bar is present, the panel name is displayed indented one space immediately below the action bar. If there is no action bar, the name is indented one space in the left-hand corner of line 1 on the panel.

When SDF2/DM is initially entered, the panel name is off.

PFSHOW [ON|OFF]
Toggles the display of the function key area. The first time you enter the PFSHOW command without parameters, the keys are removed from the display. If you enter the command again, the function key area is displayed.

PFSHOW ON displays the function key area.

PFSHOW OFF specifies that the function key area is not to be displayed.

PRINT
Records a snapshot of the physical screen image for subsequent printing.

RETRIEVE
Displays a previously issued command in the command field. For more information, see “Using the RETRIEVE command” on page 221.

RIGHT
Scrolls right. For more information, see “Using the scrolling commands” on page 222.

UP
Scrolls toward the top of the data. For more information, see “Using the scrolling commands” on page 222.

WINDOW

The WINDOW command can be used to move any pop-up window, except a message pop-up window. The top left-hand corner of the window frame is moved to the current cursor position. Therefore, when users type the WINDOW command on the command line, they must move the cursor to the required position before pressing the Enter key.

If the WINDOW command is assigned to a function key, the window is repositioned immediately to the current cursor position.

Users can also reposition a window without explicitly using the WINDOW command. They do this as follows:

1. Place the cursor anywhere on the active window frame, then press Enter.

   SDF2/DM interprets this as an implicit WINDOW command request and displays the message `WINDOW MOVE PENDING`.

2. Move the cursor to where they want the top left-hand corner of the window frame, and press Enter a second time.

If the window does not fit on the physical screen at the specified location, it is repositioned to fit on the screen.

The WINDOW command cancels a message pop-up window (see "How message pop-ups are displayed" on page 115).

Tutorial commands

Within the tutorial, the following commands can be used. The minimum abbreviation for each command is shown in boldface text.

RIGHT or press Enter	Displays the next sequential page or, if the cursor is in a scrollable area, scrolls down.
BACK or **LEFT**	Returns to the previously viewed panel or, if the cursor is in a scrollable area, scrolls up. SDF2/DM can remember up to 20 previous panels.
SKIP or **DOWN**	Advances to the next topic.
UP	Displays the menu that is one level up in the hierarchy.
TOC	Displays the tutorial table of contents menu.
INDEX	Displays the help index.
HELP	Displays help for help.
END or **CANCEL**	Ends the tutorial.

Using the RETRIEVE command

The RETRIEVE command displays a previous command in the command field and positions the cursor immediately after the command text. In most cases, a function key will be set to the RETRIEVE command.

SDF2/DM maintains a stack of commands, starting with the most recent command and going backward. The exact number of commands in the stack depends on the length of each command. The stack contains only text that is typed in the command field, not commands that are issued by function keys.

For example, if F12 is set to RETRIEVE, a user could do the following:

1. Type **up 15**, then press **Enter**.
2. Press **F7**.
3. Type **10**, then press **F7**.
4. Press **F12**. The number 10 appears in the command field.
5. Press **F12**. The command UP 15 appears in the command field.
6. In the command field, change 15 to **13**, then press **Enter**. The command UP 13 is processed. The stack now consists of the following commands (in order):

   UP 13
   10
   UP 15

When a command other than RETRIEVE is issued, any text in the command field is added to the top of the stack, whether or not it is a valid command. All other entries in the stack are moved down one position. If there is not enough room for the entire bottom entry, the bottom entry in the stack is dropped.

If the text in the stack is longer than the current command field, only the text that fits in the command field is displayed, and only this text will be processed if the user presses Enter. If the panel does not contain a command field, pressing the retrieve function key simply moves down the stack without displaying anything.

By repeatedly issuing the RETRIEVE command, a user can move through the stack. To return to the top of the stack, the user can press Enter with the command field blank. The next time RETRIEVE is issued, it will display the command at the top of the stack.

Using the scrolling commands

The SDF2/DM scrolling commands can be used whenever an application prototype presents more information than will fit on the current screen. The specific situations in which this occurs are as follows:

- A panel contains a scrollable area. SDF2/DM controls the scrolling.
- A panel contains a dynamic area that is scrollable. SDF2/DM passes the scroll amount and scroll direction to the application prototype, which controls the scrolling.
- The user is viewing the tutorial, which is a series of logically connected panels. The application developer specifies the logical connections between panels.

Scrolling within a scrollable area

The UP, DOWN, RIGHT, and LEFT commands scroll the data in a scrollable area.

If the cursor is within the scrollable area, data is scrolled so that the current cursor position is moved to the top, bottom, left margin, or right margin of the screen. If the cursor is outside the scrollable area, data is scrolled up or down one line less than a full page, or left or right one column less than a full page.

Scrolling within a dynamic area

The UP, DOWN, RIGHT, and LEFT commands scroll the data in a dynamic area that is scrollable. You can specify any of the following scroll amounts:

1–9999	Scrolls a number of lines (up or down) or a number of columns (left or right).
PAGE	Scrolls one page, which is the amount of information currently visible on the screen.
DATA	Scrolls up or down one line less than a full page, or left or right one column less than a full page.
HALF	Scrolls half a page.
MAX	Scrolls to the top, bottom, left margin, or right margin.
CSR	Scrolls so that the current cursor position is moved to the top, bottom, left margin, or right margin of the screen. If the cursor is not in the body of the data, or if it is already positioned at the top, bottom, left margin, or right margin, a full page is scrolled.

In a panel definition, if the first input field after the command field is exactly four characters long, this field is used as the scroll amount field. The scroll amount field or the system variable ZSCROLLD contains the default scroll amount. If no default scroll amount is specified, PAGE is used.

When a scroll request is issued, the system variable ZSCROLLA contains the scroll amount and the system variable ZSCROLLN contains the number of lines or columns to be scrolled, based on the value of ZSCROLLA. The system variable ZVERB contains the scroll direction. If ZSCROLLA is MAX, the value of ZSCROLLN is not meaningful.

For example, if the dynamic area contains 12 lines, the scroll amount field contains the value PAGE, and the user enters DOWN HALF, the system variables have the following values:

ZSCROLLD	PAGE
ZSCROLLA	HALF
ZSCROLLN	6 (half of 12)
ZVERB	DOWN

Chapter 10. System variables

The system variables are described in the following tables.

Commonly used system variables that a dialog can access are listed here. They are grouped by topic.

- The first column gives the name of the variable.
- The second column indicates the variable's type. The following abbreviations are used:

in	Input variable, set by a dialog to provide information to SDF2/DM
out	Output variable, set by SDF2/DM to provide information to dialogs
non	Nonmodifiable output variable
i/o	Both an input and an output variable

- The third column gives the length of the variable.
- The fourth column gives a brief description of the variable.

Numeric system variables set by SDF2/DM are right-justified and padded with zeros on the left, if necessary. If a program function uses the VCOPY service to access the variable, the value will be in character string format rather than in fixed binary format.

All system variables are stored in the shared pool.

Figure 53. Time and date variables

Name	Type	Length	Description
ZDATE	non	8	Current date (format mm/dd/yy)
ZDAY	non	2	Current day (format dd)
ZMONTH	non	2	Current month (format mm)
ZSTDYEAR	non	4	Current year (format yyyy)
ZYEAR	non	2	Current year (format yy)
ZSTDDATE	non	10	Current date (format yyyy/mm/dd)
ZIDATE	non	8	Current date (format yy/mm/dd)
ZSTDJUL	non	8	Current Julian date (format yyyy.ddd)
ZJUL	non	6	Current Julian date (format yy.ddd)
ZSTDTIME	non	8	Current time (format hh:mm:ss)
ZTIME	non	5	Current time (format hh:mm)

Figure 54. General variables

Name	Type	Length	Description
Z	non	0	Null variable
ZAPPLID	non	8	Name of the application prototype
ZCMD	i/o	n/a	Command line input
ZDBCS	non	3	DBCS terminal indicator (YES or NO)
ZENVIR	non	32	Environment description: VSE plus version.release.mod. For example: VSE 1.6.0
ZGBORDER	i/o	3	YES if graphic characters are used for the action-bar separator, pull-down window frames, and pop-up window frames.
ZGE	non	3	Graphic escape support indicator (YES or NO)
ZHOSTCMD	i/o	3	Enables or disables host command support (YES or NO)
ZKEYHELP	in	8	Keys help panel identifier. If a keys help panel is not specified in the referenced keylist, the application prototype can provide the keys help panel name in this variable. If the help panel name is present as part of the referenced keylist definition, it takes precedence over the ZKEYHELP value. This system variable must be set each time the keys help panel is changed.
ZPDC	i/o	2	Pull-down choice selection number
ZPLACE	i/o	7	Command line placement (ASIS or BOTTOM). BOTTOM moves the command line to the bottom of the panel, if it was coded within the first six lines of the panel in the panel definition. A text line that follows the command line is assumed to be the message line and is moved with the command line. ASIS leaves the command line in the position in which it appears within the panel definition.
ZUSER	out	8	User ID
ZVERB	out	8	Command verb (UP, DOWN, LEFT, RIGHT, RFIND, HELP, EXIT, or CANCEL)
ZWINTTL	in	n/a	Title to be displayed in pop-up window frame

Figure 55. Terminal and PF key variables

Name	Type	Length	Description
ZPFSHOW	i/o	4	PFSHOW command status (ON or OFF)
ZPFxx	i/o	80	Variables whose value is set by the KEYS command. If there is no keylist, ZPF01 is the value of function key 1, ZPF02 is the value of function key 2, and so on. Any function key from 13 through 24 always has the same effect as the corresponding function key from 1 through 12. If there is a keylist, the values of ZPF01 through ZPF12 have effect only if you use these variables in your keylist.
ZSCREEND	non	4	Screen depth available for dialog use.
ZSCREENW	non	4	Screen width available for dialog use. ZSCREEND and ZSCREENW are the dimensions of the physical display screen.
ZTERM	out	8	Terminal type (3270 or 3270KN). Use 3270KN for a terminal that supports Japanese Katakana characters. With 3270KN, all text is translated to uppercase.

Figure 56. Scrolling variables

Name	Type	Length	Description
ZSCROLLA	out	4	Value from scroll amount field (PAGE, MAX, HALF, CSR, or a number)
ZSCROLLN	out	4	Scroll number as computed from the value in the scroll amount field
ZSCROLLD	in	4	Default scroll value for dynamic areas that are scrollable

Figure 57. Message variables

Name	Type	Length	Description
ZERRALRM	out	3	Alarm indicator (YES or NO)
ZERRHM	out	8	Help panel associated with the message
ZERRLM	out	80	Long message text
ZERRMSG	out	8	Message ID
ZERRSM	out	24	Short message text
ZERRTYPE	out	8	Message type
ZERRWIND	out	6	Message window type

Figure 58. Help panel variables

Name	Type	Length	Description
ZCONT	i/o	8	Name of next continuation panel.
ZHELPTTL	i/o	n/a	Title to be displayed in a help pop-up window frame.
ZHINDEX	i/o	8	Name of first index panel.
ZHTOP	i/o	8	Name of the main tutorial menu.
ZIND	i/o	4	YES specifies an index page; SCRL specifies a scrollable index panel.
ZSKIP	i/o	8	Name of the next skip panel.
ZTUTOR	i/o	3	Set to YES if the user is running the tutorial.
ZUP	i/o	8	Name of parent panel.

Figure 59. Selection panel variables

Name	Type	Length	Description
ZPARENT	i/o	8	Parent menu name (when in explicit chain mode).
ZPRIM	i/o	3	YES specifies the panel is a primary selection panel.
ZSEL	i/o	8	The command input field truncated at first period.
ZSELRC	i/o	8	The return code from a selected program or command.

Figure 60. File-tailoring variables

Name	Type	Length	Description
ZTEMPF	out	17	The member name (*member.type*) of the file-tailoring output
ZTEMPN	out	16	The sublibrary (*lib.sublib*) of the file-tailoring output

Appendix. Special characters

The following characters have special meanings within a panel definition:

Quotes Text can be enclosed in single quote (') characters. It *must* be enclosed in single quotes if it contains any of the following characters:

```
blank < ( + | ) ; ¬ - , > : =
```

To indicate a null field, type two single quotes:

```
''
```

To indicate a single quote, type two single quotes within a pair of single quotes:

```
'O''CLOCK'
```

For each additional single quote, double the number of single quotes that you type.

Ampersand If an ampersand (&) is followed by a blank space or by the end of a line, it is interpreted as the ampersand character. Otherwise, an ampersand character indicates that the text that follows it is a variable name. The variable name ends with the first nonalphanumeric character (for example, a blank, period, or comma).

When a panel is displayed, the variable name and the preceding ampersand are replaced by the current value of the variable with all trailing blanks stripped. (This means that a variable whose value is a string of blanks is equivalent to a variable whose value is null.)

To indicate an ampersand character that is not followed by a blank or the end of the line, type two ampersand characters:

```
A&&B
```

SDF2/DM makes only one pass when replacing variable names. This means that the previous example will have the value A&B and that this value will not later be replaced by A plus the value of variable B. You cannot construct variable names out of variable values in SDF2/DM.

Period A period (.) at the end of a variable name indicates that the variable is to be concatenated with the following text without an intervening blank. To produce a variable value followed by a period followed by text, specify two periods instead of one.

These rules are illustrated in Figure 61. The first column contains the name of a variable. The second column contains an assignment statement for the variable. The third column contains the value of the variable after the assignment statement is processed. The fourth column explains why the variable has this value.

Figure 61. How special characters are interpreted

Variable	Assignment	Value	Explanation
A	`&A=''`		A pair of single quotes enclose a null (blank) field.
B	`&B=''''`	`'`	Two single quotes inside a pair of single quotes are translated to one single quote.
C	`&C='O''CLOCK'`	`O'CLOCK`	The two single quotes are translated to one single quote.
D	`&D='49''40'''''`	`49'40''`	The two single quotes are translated to one single quote; the four single quotes are translated to two single quotes.
E	`&E=&B`	`'`	The value of the variable B, which is one single quote.
F	`&F='A&B'`	`A'`	The letter A followed by the value of the variable B.
G	`&G='A&&B'`	`A&B`	The letter A followed by an ampersand followed by the letter B.
H	`&H='&G X'`	`A&B X`	The value of the variable G (A&B) followed by a space followed by the letter X.
I	`&I='&G.X'`	`A&BX`	The value of the variable G followed by the letter X with no intervening space.
J	`&J='&G..X'`	`A&B.X`	The value of the variable G followed by a period followed by the letter X.
K	`&K='A..B'`	`A..B`	The letter A followed by two periods followed by the letter B.

Line continuation

A list can continue on the next line. In this case, the end of a line is equivalent to a comma or blank in separating list items.

If you must split a literal value, use a plus sign at the end of the line as a continuation character. The plus sign and all intervening blanks are replaced by one blank space.

For example, the following are all equivalent:

```
TRANS (&XYZ 1,'CASE ONE' 2,'CASE TWO')
TRANS (&XYZ   1   'CASE ONE'   2    'CASE TWO')
TRANS (&XYZ   1   'CASE ONE'
   2    'CASE TWO')
TRANS (&XYZ 1 'CASE    +
         ONE' 2,'CASE TWO')
```

Double-byte characters

Double-byte character set strings must be enclosed within shift-out (hexadecimal 0E) and shift-in (hexadecimal 0F) characters.

For example:

```
'SSSS[DBDBDBDB]SSS'
```

Single-byte characters are represented here by an S, double-byte characters by DB, and the shift-out and shift-in characters by brackets ([ ]).

Glossary of terms and abbreviations

Glossary terms are defined as they are used in this book. Some definitions have been taken from *American National Standard Dictionary for Information Systems*, in which case they are marked with (A); other definitions are from the *Information Technology Vocabulary*, in which case they are marked with an (I). Definitions without source labels are IBM definitions. If you cannot find the term you are looking for, refer to the index, the online reference index, or to the *IBM Dictionary of Computing,* SC20-1699.

A

abend. Abnormal end of task.

action bar. In Common User Access architecture, the area at the top of a window that contains choices that give a user access to actions available in that window.

action bar choice. A textual item on an action bar, which provides access to menus that contain choices that can be applied to an object.

active partition. The partition that contains the cursor. It can be scrolled vertically. While a partition is active, the cursor “wraps around” at the viewport boundaries, and the *ENTER* key (or input key) transmits data from that partition only.

adjunct. An optional field in the data structure that is added to a field in the data structure, which contains data to be displayed. It enables the application program to vary a specific presentation attribute (or set of attributes) at run time.

AID. See attention identifier.

AID table. A table that assigns values to actions performed by the user of the application program. An action may be, for example, the pressing of a program function key. The values are used by the application program.

APAR. Authorized program analysis report.

application. A collection of software components used to perform specific types of user-oriented work on a computer. Typical examples are payroll applications, airline seat-reservation systems, and stock-control systems.

application attribute. A property of a variable field, such as justification of data in the data structure. Contrast with presentation attribute.

application development (AD). The defining, writing, and testing of a program for a specific solution or application problem.

application element. Any single item in the data structure.

application prototype. A simulation of an application by presenting some or all panels used in the application in a predefined order. See operational prototype and simulative prototype.

area. A rectangular part of a format, whose contents (text or graphics) are provided at run time by the application program. See graphic area and dynamic area.

area attribute. An attribute that affects the properties of an area. It can be, for example, extendable or scrollable.

area mark. A mark used to define an area (see area), such as a graphics area or a dynamic area.

array. A named, ordered collection of variable fields, all of which have identical names and attributes. An array has a specified occurrence number denoting the number of elements in the array. See horizontal array and vertical array.

array index. A number in parentheses that appears next to the name of an array. For example, in the name of the element *a(3)* of the array *a*, *3* is the array index.

assembler (ASM). A computer program that converts assembly language instructions into object code.

attention identifier (AID). A character in a data stream indicating that the user has pressed a key, such as the Enter key, that requests an action by the system.

attribute. See presentation attribute and application attribute. See application attribute, area attribute, background attribute, character attribute, field attribute, inherent attribute, and presentation attribute.

attribute descriptor. A symbol that denotes a set of attributes.

attribute line. A line showing the attribute descriptors assigned to the field.

autosave. An automatic save facility in which the user can define a specific number of alterations after which a temporary save occurs automatically.

autosave library. A library in which the saved objects are stored.

B

background attribute. The attributes associated with background text.

background text. All text on a panel that is not within a constant or variable field.

base name. The name that is used in a based data structure as a pointer variable that identifies the location of the data.

block. In SDF II, a rectangular part of a format that is defined by the position command for such commands as moveblock or delblock.

C

C. A high-level programming language.

character attribute. An attribute that applies to a single character.

CICS/BMS. Customer Information Control System/Basic Mapping Support.

COBOL. A high-level programming language, based on English, that is used primarily for business applications.

Common User Access (CUA) architecture. Guidelines for the dialog between a person and a workstation or terminal.

constant field. In SDF II, a field that contains constant text, which has attributes that differ from background attributes. Contrast with variable field.

control table. (1) In IMS/MFS, a user-defined table of operator control functions; a specific control function is invoked when the input device data or data length satisfies a predefined condition. (2) In SDF II, an object that corresponds to an operator control table in IMS/MFS.

conversion. A process by which an object defined for a specific target system is changed so that it becomes an object for another target system. The converted object will retain those properties which are supported by the new target system.

Cross System Product (CSP/AD and CSP/AE). A set of licensed programs designed to permit the user to develop and run applications using independently defined maps (display and printer formats), data items (records, working storage, files, and single items), and processes (logic). The Cross System Product set consists of two parts: Cross System Product/Application Development (CSP/AD) and Cross System Product/Application Execution (CSP/AE).

CSP/AD. Cross System Product/Application Development.

CSP/AE. Cross System Product/Application Execution.

CUA. See Common User Access architecture.

CUA attribute. Synonym for CUA panel element attribute.

CUA panel element. The smallest named part of a panel, such as a title, which is based on CUA architecture.

CUA panel element attribute. In SDF II, any attribute associated with a CUA panel element type. Synonymous with CUA attribute.

CUA panel element type. In SDF II, used as a reference to a class of CUA panel elements. Synonymous with CUA type.

CUA type. Synonym for CUA panel element type.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

D

DASD. Direct access storage device.

data mark. Synonymous with DATAIN/DATAOUT attribute characters in ISPF.

data structure. In SDF II, a structure that is part of a panel. For output, it describes how data is provided by the application. For input, it describes how data is presented to the application.

DBCS. Double-byte character set.

device list. A list of compatible device types. It is defined by the system programmer.

device table. Synonym for device type table.

device type. In SDF II, the name of a device or of a device list.

device type editor. An editor used for creating and maintaining the device type table.

device type table. A table containing the names of all device types supported by SDF II, together with the features available on the devices. It is maintained by the SDF II administrator.

DFLD. Device field.

dialog. (1) The interaction between a user and a computer. (2) In SDF II, one or more panels and associated logic that establish an interactive session between SDF II and a user. A dialog prompts the user to enter information appropriate to the function requested and displays the results.

direct access storage device (DASD). A device in which access time is effectively independent of the location of the data. (A)

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS.

DSECT. Dummy control section.

dummy control section (DSECT). A control section that an assembler can use to format an area of storage without producing any object code. (A)

dynamic area. In SDF II, an area that is filled with text at run time by the application program.

E

EBCDIC. See extended binary-coded decimal interchange code.

emphasis class. In SDF II, a set of predefined attributes. Emphasis classes can be specified for fields, marks, and attribute descriptors.

EXEC. An executable procedure that contains operating system commands and execution control statements.

extended attribute. Any one of the color, highlight, programmed symbol set, outlining, mixed, or validation attributes.

extended binary-coded decimal interchange code (EBCDIC). A coded character set of 256 8-bit characters.

extended external source format. In SDF II, an extension of CSP/AD's external source format representing certain properties of CICS/BMS and IMS/MFS. See external source format.

external source format. CSP/AD's external source format is a commonly used means of representing applications and panels in an AD/Cycle framework. The format consists of a readable syntax of mark-up tags and attributes.

F

field attribute. A defined characteristic of a field, such as protected or unprotected, alphanumeric or numeric, detectable or nondetectable, displayable or nondisplayable, or intensity. See presentation attribute and inherent attribute.

field format. A field property that determines the character set that can go into a given field.

format. A format is part of a panel. It defines how data appears on a screen. For output, it defines how data is presented on a screen. For input, it defines how data is entered on a screen by a user. A format may consist of different definitions for different device types. These definitions are called format instances.

format element. A part of a format, such as a variable field, a constant field, a dynamic area, a graphic area, a repeat format, or an include panel.

format instance. A part of a format that defines the appearance of data for a particular device type.

format mode. One of the four modes in which SDF II can display the layout of a panel. In this mode, marks show the extent of fields and areas. Contrast with initial value mode, name mode, and sample value mode.

G

GDDM–IMD. Graphical Data Display Manager — Interactive Map Definition.

generation. In SDF II, a process by which objects are created for use in the target systems or for prototyping the application.

graphical data display manager (GDDM). A group of routines that allows pictures to be defined and displayed procedurally through function routines that correspond to graphic primitives.

H

horizontal array. An array that is read from left to right and line by line. For example:

choice (1)	choice (2)
choice (3)	choice (4)

See array and vertical array.

I

import. In SDF II, a process by which objects are imported into SDF II from one of the supported target systems, from SDF/CICS, or from an external source format structure.

IMS/MFS. Information Management System/Message Format Service.

include panel. A panel that is included in another panel. Examples are headers and trailers.

Information Management System/Virtual Storage (IMS/VS). A database/data communication (DB/DC) system that can manage complex databases and networks. Synonymous with IMS.

inherent attribute. An attribute that can be defined for variable and constant field marks, and data marks. After the field is defined, inherent attributes cannot be changed.

initial mode. In SDF II, one of the four modes in which SDF II can display the layout of a panel. In this mode, the Format window shows each initial value in its variable field. Contrast with format mode, name mode, and sample mode.

initial value. A value the SDF II user assigns to a variable field. The application program displays this value at run time if no value has been provided by the application.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogs between the application programmer and terminal user. (A)

ISPF. Interactive System Productivity Facility.

ISPF/PDF. Interactive System Productivity Facility/Program Development Facility.

M

mark. In SDF II, a character used to define a format element, such as a field or area, or to provide some editing function. Examples include area marks, character marks, separator marks, and spacer marks.

MRI. Machine-readable information.

machine-readable information (MRI). All textual information contained in a program, such as a system control program, an application program, or microcode. MRI includes all information that is presented to or received from a user interacting with a system. This includes menus, prompts, messages, report headings, commands, and responses. MRI may appear on printers or on display panels. (A)

MVS. Multiple virtual storage. Implies the MVS/XA product and the MVS/ESA product.

N

name mode. One of the four modes in which SDF II can display the layout of a panel. In this mode, the Format window shows the name of the variable field in the field. Contrast with format mode, initial mode, and sample mode.

national language support (NLS). The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, MRI, or documentation of a product. (A)

NLS. National language support.

nonprogrammable terminal (NPT). In Basic Common User Access architecture, a terminal attached to a host processor in which most of the user-interface functions are controlled by the host processor. (A)

NPT. Nonprogrammable terminal.

O

object. In SDF II a panel, panel group, partition set, or AID table stored in an SDF II library.

Figure 62 shows the equivalents for these objects in the target systems.

Figure 62. SDF II objects and target system equivalents

SDF II Object	IMS/MFS	CICS/BMS [1]	GDDM–IMD	CSP/AD or VisualGen	ISPF
Panel	Format set	Map	Map	Map	Panel
Panel group		Map set	Map group	Map group	
Partition set	Partition definition block	Partition set			
AID table	PF key parameter of the DEV statement		AID table		
Control table	Operator control table				

[1] SDF/CICS uses the same terms as CICS/BMS

operational prototype. A simulation of an application program to test or review simple functions, such as simple database access, scrolling, error reporting, and online help panels. For some application programs, an operational prototype may include some characteristics of a database, including some program code or SDF II dialog manager tables. The operational prototype is used to determine the needs of the user of the application program and to ensure that the application program meets those needs. See simulative prototype.

P

page. In SDF II, part of a format instance that corresponds to an IMS/MFS physical page.

panel. (1) The information that is displayed at any one time on the screen. (2) An SDF II object. It consists of formats, data structures, and various tables. Each panel has at least one format.

panel command. A command that affects a part of the panel, the whole panel, or the flow of SDF II. Panel commands are entered on the command line. They can be assigned to program function keys.

panel element. (1) An element of a panel as displayed in the Define Panel Instances dialog, which denotes one of the following:

- Format
- Format instance
- Page
- Data structure

(2) A line in the Specify Panel Elements dialog. It is used by the panel construction utility to create one or more fields, panel text, or a repeat format on the panel to be constructed.

panel group. An object within SDF II that contains a list of panel names and describes the properties of these panels.

panel group instance. A part of a panel group that describes the properties of the panel group for a particular device type.

partition. All or a portion of the screen. Data is presented within the partition through a viewport, which is defined when the partition is created.

partition set. An SDF II object that consists of a group of partitions designed to share the same screen.

partition set instance. A part of a partition set that describes the properties of the partition set for a particular device type.

PL/I. A programming language that is designed for use in a wide range of commercial and scientific computer applications. (A)

presentation attribute. An attribute that defines how information is presented on the screen, such as highlighting and color. Contrast with application attribute.

program temporary fix (PTF). A temporary solution or bypass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program.

prototype. See simulative prototype and operational prototype.

PTF. Program temporary fix.

pull-down. In Common User Access architecture, a list of choices associated with a choice on the action bar. A user selects a choice from the action bar and a pull-down menu appears.

pull-down choice. A textual item on a menu. A user selects a choice to work with an object in some way.

R

reference name. A 1- or 2-character name used by SDF II as a synonym for the name of a variable field.

repeat format. A rectangular part of the format that can be repeated down a panel. All instances of a repeat format must have the same variable fields at the same relative horizontal positions as in the source format.

Report Program Generator II (RPG II). A commercially oriented programming language specifically designed for writing application programs intended for business data processing. (A)

Restructured Extended Executor (REXX). An interpretive language used to write command lists.

REXX. Restructured Extended Executor language.

RPG II. Report Program Generator II.

S

sample mode. One of the four modes in which SDF II can display the layout of a panel. In this mode, the Format window shows each sample value in its variable field. Contrast with format mode, initial mode, and name mode.

sample value. A value the SDF II user assigns to a variable field. SDF II displays this value when the panel is tested or during prototype simulation.

Screen Definition Facility/Customer Information Control System (SDF/CICS). An online application development tool used by application programmers to define or edit maps, map sets, and partition sets for CICS/VS Basic Mapping Support. (A)

Screen Definition Facility II (SDF II). An interactive application development tool that helps application developers to define, maintain, import, and generate screen objects, such as panels, panel groups, partition sets, attention identifier (AID) tables, and control tables, as appropriate, for its target systems.

scrollable area. The window in the main panel behind which a scrollable area format can be scrolled.

scrollable area format. A separate format used with a scrollable area.

SDF/CICS. Screen Definition Facility/Customer Information Control System.

SDF II. Screen Definition Facility II.

SDF II dialog manager. The dialog management component of Screen Definition Facility II.

SDF2/DM. SDF II dialog manager.

separator. In SDF II, a mark used to separate the length of a field, its name, and its mark.

shift-in character (SI). A code extension character used to terminate a sequence that has been introduced by the shift-out character to make effective the graphic characters of the standard character set. (I) Contrast with shift-out character.

shift-out character (SO). In SDF II, a code extension character that substitutes, for the graphic characters of the standard character set, DBCS. Contrast with shift-in character.

SI. Shift-in character.

simulative prototype. A simulation of a series of panels used by an application program to test or review the primary flow of interactions between the application program and its users. The panels may display initial values and may accept data entered by a user. See operational prototype.

skeleton. An object used as a model when creating a new object.

skip after attribute. In SDF II, a presentation attribute that causes the cursor to skip to the next unprotected field when the field in which the cursor is located has been filled.

SO. Shift-out character.

spacer. In SDF II, a mark that positions information on lines during panel definition; it is typically used for centering.

specification object. Synonym for object.

T

target system. A system under which the application using an SDF II panel can be run. For example, CICS/BMS, CSP/AD, VisualGen, ISPF, GDDM–IMD, and IMS/MFS.

U

user exit routine. A user-written routine that receives control at predefined user exit points. In SDF II VSE, for example, it is an EXEC.

V

variable field. A field in which data may be changed by the application program or by the user. It has a character string, which can be empty, defined at run time as its contents. If no contents are provided at run time, the initial value, if defined at specification time, is taken as default instead. Contrast with constant field.

vertical array. An array that is read from top to bottom and column by column. For example:

choice (1)	choice (3)
choice (2)	choice (4)

See array and horizontal array.

Virtual storage extended (VSE). An IBM licensed program whose full name is the Virtual Storage Extended/Advanced Function. It is a software operating system controlling the execution of programs. (A)

VSE. Virtual storage extended.

W

window. In SDF II, a rectangular part of the screen where scrollable data is displayed and can be manipulated.

SDF II publications

The SDF II Release 6 publications are:

SDF II Licensed Program Specifications, GH12-6318
: Contains the product specifications and warranty information.
Audience: Data processing manager, system programmer.

Introducing SDF II Release 6 for VSE, GH12-6314
: Summarizes the functions, uses, requirements, and advantages of SDF II.
Audience: Data processing manager, system programmer.

SDF II General Introduction, SH12-6315
: Introduces SDF II to new users and explains how to define simple panels. It also explains how to prototype the flow of panels and main functions of an application.
Audience: System programmer, application programmer, application user.

SDF II Primer for CICS/BMS Programs, SH12-6313
: Explains how to use SDF II to develop objects for applications that run under CICS/BMS.
Audience: System programmer, application programmer, application user.

SDF II Run-Time Services, SH12-6312
: Provides a comprehensive reference to the language and functions of the SDF II dialog manager (SDF2/DM).
Audience: System programmer, application programmer.

SDF II Administrator's Guide, SH12-6311
: Describes how to customize SDF II on a VSE system. It also explains how to import objects into SDF II, how to set up and work with libraries, how to run SDF II from batch, and how to identify and report problems in SDF II to IBM support personnel.
Audience: System programmer, application programmer.

SDF II Reference Summary, SX12-5012
: Lists and explains SDF II line and panel commands. It also lists the main dialogs and functions of SDF II.
Audience: System programmer, application programmer, application user.

Index

Your comments, please ...

Screen Definition Facility II for VSE
Run-Time Services
Release 6

Publication No. SH12-6312-00

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you mail this form to us, be sure to print your name and address below if you would like a reply.

You can also send us your comments using:

- A fax machine. The number is: +43–1–21145–4490
- Internet. The address is: kpesen@vnet.ibm.com.

  Please include the title and publication number (as shown above) in your reply.

Name	Address
Company or Organization	
Phone No.	

Your comments, please ...
SH12-6312-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Vienna Software
SDF Development/Service
Department 00/174
Lassallestrasse 1
A–1020 Vienna
Austria
Europe

Fold and Tape **Please do not staple** Fold and Tape

SH12-6312-00

Cut or Fold
Along Line

Program Number: 5746-XXT

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH12-6312-00