software AG

Com-plete

Application Programming

Version 6.8.2

July 2023

ADABAS & NATURAL

Table of Contents

Application Programming

This documentation is a reference guide for Com-plete application programmers. It contains all the information necessary to write online and batch programs in the Com-plete environment.

Com-plete provides a wide range of functions for the application programmer. The description of each Com-plete function in this documentation follows the same pattern. Each description consists of:

- A general description of the function;
- A description of the function statement format according to the **syntax convention**;
- Return codes pertinent to the function;
- Possible abnormal terminations and their cause(s).

The Com-plete application programming information is organized as follows:

Introduction to the API	Introduction to application programming using Com-plete functions
Terminal Functions and Paging	Describes functions related to terminals and terminal paging.
Storage Access Functions and Task Management	Describes functions relating to access to external storage systems (including Adabas), as well as task managment functions.
Message Switching and Printout Spooling	Describes functions relating to message switching and printout spooling functions (including NSPOOL).
Miscellaneous Functions	Describes miscellaneous functions which enable the application program to utilize a wide range of features.
Function Tables	Presents various control blocks and code tables.

1 API Conventions

Standard conventions are used throughout this documentation in the descriptions of the various Com-plete functions.

Syntax Symbols

In the descriptions of function statement format, arguments enclosed in square brackets, [ ], are Optional.

However, an argument enclosed in square brackets can only be omitted if it is the last argument in the argument list. Com-plete handling requires that an optional argument which is not the last argument in the list *must* still be included. In this situation, the argument name should reference a field that contains zeros or blanks, depending upon whether the field is numeric or alphanumeric.

The following format statement illustrates the use of square brackets to indicate optional arguments.

```
function (retcode,argument1,
      [,argument2]
      [,argument3])
```

This means that *argument3* can be omitted, and if *argument2* is not required, you must give it a dummy value as described above.

If two or more options enclosed within a single set of brackets are separated by a vertical bar, |, only one of the options must be coded.

Curly braces, {}, indicate that one of the enclosed items separated by a vertical bar(s), |, is mandatory.

The following format statement illustrates the use of curly braces and vertical bars to describe mandatory items.

```
WRT {C|D|R}(....)
```

This means that the choice of function is determined by the specified suffix (enclosed within braces) to the character string WRT (that is, function WRTC, WRTD, or WRTR).

> **Note:** Do not type square brackets, curly braces or vertical bars as part of the function statement.

Argument Coding

Com-plete function names and their corresponding arguments are used to specify the services and options to be performed. The following general rules summarize the coding conventions followed throughout this documentation:

Rule 1: You code CALL statements to a subroutine with the function name.

Rule 2: Arguments are positional and must be coded in the order given.

Interface Linkage

All programs, except Natural, which use Com-plete functions must be linked with the appropriate interface routines.

Online applications:
Online applications must be linked with the Com-plete interface modules which were loaded to a data set during Com-plete installation. There is a separate interface module for each individual function.

Batch applications:
Batch applications must be linked with the module COMPBTCH, supplied on the Com-plete load library, which contains entry points for all Com-plete functions allowed from a Batch environment.**Com-plete Functions For Batch And Online Programs** lists the Com-plete functions which are available from Batch.

For further information on how to run Batch programs, refer to the Batch section of the Com-plete System Programming documentation.

Return Codes

Most Com-plete functions cause a return code to be issued at completion of their operation. The return code is placed in the first argument of the CALL statement.

Return codes supply information to the application program that can be used to determine the further course of program execution. For example, the following return codes are given in response to a terminal read request:

0	Data read equals that requested.
4	More data is available to read.
8	Less data was read than requested.

2   About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format `folder.subfolder.service`, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
		Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the	symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to "Developer Center", "User Center" or "Documentation".

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.

- Ask questions and find answers in our discussion forums.

- Get the latest Software AG news and announcements.

- Explore our communities.

- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.

- Search the Knowledge Center for technical information and tips.

- Subscribe to early warnings and critical alerts.

- Open and update support incidents.

- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

I Introduction to the API

Application programs perform Com-plete functions by calling Com-plete subroutines from the distributed subroutine library.

High level languages such as Natural, COBOL, PL/I, and FORTRAN use a CALL statement. Assembler programs can use the CALL macro provided by the operating system or an operating system-independent CM$CALL macro distributed with Com-plete.

COBOL, PL/I, FORTRAN, and Assembler programs are link edited with routines from the distributed library.

In all cases, the subroutines use standard linkage conventions.

This part of the Application Programming documentation introduces you to the way Com-plete handles user programs, points to some language-specific considerations, and describes the API.

This information is organized under the following headings:

Application Programming in Com-plete
The Application Programming Interface

3 Application Programming in Com-plete

How Com-plete Works

Online programs are initiated by request of a terminal user or by request of an executing online program. The terminal user must type

```
*pgmname  parameters
```

or invoke a program via the COM-PASS menu. In either case, *pgmname parameters*, including any terminal-dependent control characters, is placed into a terminal buffer associated with that terminal. Com-plete searches for the load module *pgmnam*, allocates a thread, places the program into the thread (if the program is not defined as RESIDENTPAGE or does not reside in the LPA/SVE), and then passes control to that program. This root program can use those Com-plete functions available to it. The root program terminates either by calling a Com-plete termination function (EOJ or WRTxD) or by returning via the language-specific return mechanism (e.g., stop run, go back, return, etc.).

Threads

During execution, online programs reside in areas called threads. User threads are fixed in size and range from a minimum of 8K to a maximum of 1008K below the 16 MB line. For each thread, a fixed-size extension can be allocated above the 16 MB line. Note that load modules linked with OVERLAY cannot be used with Com-plete.

Rolling Mechanism

Com-plete uses its own paging mechanism. Whenever an application program (or Com-plete utility) running in a user thread issues a request for a terminal I/O operation, ADABAS or ROLOUT function, the Com-plete paging mechanism can be invoked. If another user is waiting for the thread, the entire contents of the thread are paged out (roll-out) to the roll buffer and the waiting application is paged in (roll-in). Because of this mechanism, each terminal can use its own copy of a program and alleviate the need for writing re-entrant programs.

Note that during a ROLOUT operation, the program/thread counters CPU time, REAL time and Adabas calls are reset.

COM-PASS Considerations

All programs that reside in the Com-plete program library can be invoked by the terminal user. The following points should be considered to ensure that programs are not used by unauthorized users:

- Main programs should be protected by the Com-plete Security System;

- A subprogram that is COLOADed will not be security protected;

- Any routine that is invoked from a terminal will always contain an '*' at the start of the input data buffer;

- COM-PASS checks the integrity of transactions only if the data buffer that is transferred with the transaction begins with an asterisk (*);

- A main program that fetches or attaches another should transfer a data buffer that does not start with an asterisk (*) in order to distinguish it from a terminal call.

COM-PASS allows up to nine transactions per user to be defined under the control of transaction security. This assumes that the User Profile definition option 'ALLOWED NON-MENU PRO-GRAMS' is specified as 'NO'.

Write Conversational Calls

Each transaction within COM-PASS can be suspended whenever the transaction contains a write conversational (WRTC) call to Com-plete. COM-PASS provides a return code of 16 when the transaction is restarted. To ensure the correct use of the COM-PASS Restart/Suspend facilities, each transaction that contains a write conversational call should check for an RC=16. Programs that use writes without erase should rebuild the entire screen whenever the program has been suspended.

Specific Programming Language Considerations

Each argument in a call to Com-plete must be defined with the data type required for that argument. The data types used are:

Alphanumeric	Fixed-length fields representing character data. Natural - (An). COBOL - PICTURE X(n). PL/I - CHARACTER (n). Assembler - DC CLn' '.

Binary fullword	Four-byte fields representing a signed binary value. Natural - (B4). COBOL - PICTURE S9(8) COMP. PL/I - FIXED BINARY (31). Assembler - DC F'n'.
Binary halfword	Two-byte fields representing a signed binary value. Natural - (B2). COBOL - PICTURE S9(4) COMP. PL/I - FIXED BINARY(15). Assembler - DC H'n'.

Natural

In general, Natural provides Com-plete functions to the programmer that are built into the Natural language. For example, terminal I/O functions of Com-plete are implemented via the NATURAL INPUT verb. Applications written in Natural should use the built-in facilities of Natural wherever possible.

For access to a Com-plete function not directly available in the Natural language, code a CALL statement to the Com-plete subroutine with the function's name (for example, SDOPEN for SD file open). The Natural CALL statement is equivalent to the function of COLINK; therefore, COLINK should never be called from Natural.

Com-plete functions are available to the Natural CALL statement, if those subroutines are cataloged in the Com-plete program library or are placed into the RESIDENTPAGE portion of Com-plete.

COBOL

z/OS COBOL programs should be compiled with options 'NOSTAE, NOSTATE, and ENDJOB'.

z/VSE COBOL programs should be compiled with control card:

```
CBL NOSTXIT,NOSTATE,NOCOUNT
```

and must not use LFOW or SYMDMP.

COBOL II

COBOL II enables you to generate reentrant code for the COBOL programs. This facility can be used to its optimum effect with Com-plete. Programs that are reentrant can run as RESIDENTPAGE programs within the Com-plete nucleus. Com-plete can also handle programs that run above the 16 MB line. Also, the COBOL run time routines can be linked into one COBPACK which can also be RESIDENTPAGE or in LPA. Com-plete can support this both below and above the 16 MB line as applicable.

You must choose the IGZTUNE parameters that these programs use when running under Com-plete. In particular the INIBLOW and the INIABOV parameters, which indicate the amount of storage COBOL's storage management routines will receive at startup before doing anything else. As this storage is allocated from thread, at least this amount must be available in the thead before the COBOL program can run. If this storage is never used, then valuable thread and rollbuffer space will be wasted. We therefore recommend that these are set to reflect the storage usage of the most of your COBOL programs (90% or more) and let COBOL request the storage for the rest of the programs. This will simply result in a few more getmain/freemain requests from the COBOL programs for which the allocated space is not enough.

Care should also be taken with the INIABOV parameter, as this storage is also allocated from thread and therefore is not allocated above the 16 MB line. COBOL expects this GETMAIN to always work and so, having issued a conditional GETMAIN to get the storage, proceeds to try to use the storage which may not be there if the GETMAIN fails (as can happen under Com-plete if the catalog size for the program is not large enough). This eventually results in a storage exception abend when COBOL tries to address storage which is not addressable by your program. For this reason, we recommend that if unexplainable COBOL addressing exceptions occur, this parameter should be set to 1, the Com-plete catalog size for the program increased and the program tried once more.

PL/I

With the exception of the COLINK and COXCTL functions, Com-plete does not support PL/I dope vectors; therefore, the declaration of the entry for a Com-plete function must specify the ASM option. In addition, the EXTERNAL attribute must always be specified.

The 'NOREPORT' compile option must not be used.

The COLINK and COXCTL functions must be defined with the attributes of the target program.

The following definition is recommended:

```
DCL function ENTRY EXTERNAL OPTIONS (ASM,INTER);
DCL IRETURN FIXED BIN(31) INITIAL (0);
DCL FIELD CHAR (10) INITIAL ('CONSTANT');
CALL function (IRETURN, FIELD);
```

When calling a PL/1 subroutine from a PL/1 main program, the subroutine needs to be linked with the entry point name the same as the subroutine name. This obtains the correct entry and prevents PL/1 repeatedly setting up the run-time environment. See the *PL/1 Optimising Compiler Programmer's Guide* for further information.

FORTRAN Notes

FORTRAN routines require privileged status. Catalog FORTRAN programs with the ULIB PV operand (see the Com-plete Utilities documentation).

Assembler

CM$CALL

Assembler programmers have a choice of using the CALL macro distributed and documented for their operating system or a CM$CALL macro distributed with Com-plete. The CM$CALL macro manages operating system independent calls.

The format of the CM$CALL is:

```
CM$CALL ENTRY,PARMS
```

or

```
CM$CALL ENTRY,PLIST=
```

where:

ENTRY	Required
	Name of routine. If (R15) is used, register 15 must have the address of the subroutine.
PARMS	Optional
	A list of parameters in parentheses separated by commas or null. If a list is specified, each parameter must be a valid second operand of a load address (LA) instruction.
PLIST	Optional
	A valid second operand of a load address (LA) instruction or a parenthesized register specification.

The parameter list for the CALL will be:

1. Null, if PARMS and PLIST are both absent.

2. Generated inline, if PARMS is specified, but not PLIST.

3. Generated at a location specified by PLIST, if PARMS is specified.

4. Assumed to exist at a location specified by PLIST, if PARMS are not specified. The end-of-parameter-list indicator must be set.

Be aware that Com-plete validates the parameter list. The last item in the parameter list must have the end-of-parameter-list indicator set. The LV option must be specified in the z/OS CALL macro in order to set this indicator.

4 The Application Programming Interface

Overview

Due to the logic of the new dispatching mechanism, the Com-plete Application Programming Interface (API) had to be redesigned for Com-plete 5.1. While the internal logic of the interface has changed significantly, the usage of the interface has not. This was done to encourage users to convert their existing applications to use the new interface.

While Software AG strongly recommends that all applications be converted to use the new interface as soon as possible, in order to facilitate conversion to new Com-plete versions, a bridging mechanism has been provided which enables application programs which worked under Com-plete 4.6 to function unchanged under later Com-plete versions. However, note that this bridging mechanism costs more in terms of resources than the new interface.

Prior to this new interface being introduced, all Com-plete API calls entered the Com-plete nucleus using a pseudo Supervisor Call (SVC) which was trapped by Com-plete. Changes made in versions 4.5 and 4.6 of Com-plete have facilitated the introduction of a branch entry mechanism to the API routines. Branch entry to the nucleus is the major change which will effect currently running assembler applications programs which used the MCALL interface. Applications correctly using the High Level Language Interface (HLLI) as previously documented will not have to be changed at all as you will see later.

> **Note:** The term 'correctly' above relates to the provision of an 18F savearea to the HLLI routine. Please refer to the migration notes for Com-plete 5.1 for details of API functions that previously worked even if an 18F savearea was not provided. All API functions now require an 18F savearea.

Requirements for Using the Branch Entry Interface

To use the branch entry interface, the following must be taken into account.

- Register 13 must point to a standard 18F savearea which will be used to save and restore the application program's savearea.

- Registers 14, 15, 0 and 1 will be changed by the call to the application programming interface. Registers 2 to 13 will not be altered.

- Access Registers 1 to 15 will be returned unaltered to the application program.

Differences with the Original Interface

As you will see from the above, applications using the HLLI interface will not have to be changed as they already have taken the above requirements into account. The changes that assembler programs must take into account are as follows:

- Registers 14 and 0 were generally returned intact to the application issuing the MCALL functions. This will no longer be the case.

- Register 13 did not have to point to a standard 18F savearea when issuing an MCALL function.

Relocation Issues

Programs which are relocatable and use branch entry to the new API will have to be aware of the following.

- Register 13 will *always* be relocated as it must be located in the application's thread.

- Register 14 will be relocated if it is located inside the application's thread.

Assembler Programs Using the MCALL Interface

How to use the Branch Entry Interface

In order to use branch entry to the API, two new parameters (SAVEAREA and COMREGA) have been added to the MCALL macro to determine how it will enter the Com-plete API. Note that an unchanged program will always expand to use the SVC entry as branch entry will only be used if explicitly requested on the MCALL or if globally set using the CMOPBE macro described later.

SAVEAREA	YES/NO Default: NO (unless previously specified in CMOPBE).
	This parameter indicates whether the application has an 18F savearea available at the point in the program where the MCALL is issued. When 'NO' is specified, it indicates that no savearea is available and the old SVC entry will be generated. When 'YES' is specified, it indicates that an 18F savearea is available and pointed to by register 13. Specifying 'YES' ensures that the application program will branch enter the API. How this is achieved is determined by the COMREGA parameter described below.
COMREGA	YES/NO Default: NO (unless CMOPBE was specified previously).
	This parameter indicates whether COMREG is addressable at the point in the program where the MCALL macro is issued. It determines the manor in which the API will be branch entered and is only applicable if SAVEAREA is set to 'YES'. COMREGA=YES indicates that COMREG is addressable at the point where the MCALL is issued and a USING is active on the DCOMREG DSECT for the register pointing to COMREG (e.g. register 2). This will cause the MCALL to expand to load the address of the Com-plete API from COMREG and to branch directly to it. This saves the requirement to link an additional stub module with the assembled module.

	COMREGA=NO indicates that COMREG is not addressable and causes the MCALL to expand loading the address of the entry point TLOPENT, again only when SAVEAREA=YES is specified. This entry point is contained in the module TLOPUSER therefore, this module must be linked with the load module resulting from the assembly.

Maintaining Reentrancy

The functions to get and free storage (i.e. MCALL GETMAIN and MCALL FREEMAIN) also require a savearea to use the branch entry interface which leaves the programmer in a catch 22 situation when trying to maintain re-entrancy in an assembler program. How can storage be acquired for a savearea if no savearea exists? This is addressed through the provision of two macros called CMOPGETM and CMOPFREM which are described below. Software AG recommends that these macros are only used when absolutely necessary, i.e. to get working storage at the start of a program and free it at the end of the program. In all other cases, the standard MCALL or HLLI functionality should be used.

Mixing the Branch Entry and SVC Interfaces

It is possible to mix branch entry calls to the interface with SVC entry calls, though this is not recommended. While it is technically possible, it will lead to confusion in modules. You are recommended to totally change a module to use the branch entry interface when it is being converted.

Globally Changing MCALLs for a Module

Where a module has been written according to normal IBM standards, it will generally comply with the conditions for branch entering the API. For this case, a macro (CMOPBE) has been provided to change the default for the entire module or for entire sections of a module. CMOPBE sets global indicators to cause MCALLs following the invocation of this macro to expand based on the global specifications of the CMOPBE macro. It is possible to invoke this macro a number of times to have different options for different sections of the module, however, be aware that the MCALL will take it's defaults from the last invocation of the macro*physically* preceding it in the assembler source.

Macro Descriptions

CMOPBE - Set global indicators

Syntax:

```
CMOPBE SAVEAREA=YES/NO,COMREGA=YES/NO
```

Parameters
   For a Description of SAVEAREA and COMREGA refer to the section *How to use the Branch Entry Interface*.

CMOPGETM - Get storage
This macro will acquire storage for the requested length if it is available.

Syntax:

```
CMOPGETM LEN=length,COMREGA=YES/NO[,LOC=ANY|BELOW]
```

Parameters

LEN	Required
	Amount of required storage. (see Note 1)
COMREGA	Optional
	(See Note 3)
LOC	Optional
	Location of storage requested. The default depends on the setting of AMODE. With AMODE=24, the default is BELOW. With AMODE=31, the drfault is ANY.

Return Codes:

0	Storage gotten successfully. The address will be returned in Register 1
4	Storage unavailable in the thread.
8	Should not occur
12	Either the request is invalid or the FQE chain in the thread is corrupted.

CMOPFREM - Free storage
Free storage previously acquired by a CMOPGETM request.

Syntax

```
CMOPFREM LEN=length,ADDR=address,COMREGA=YES/NO
```

Parameters

LEN	Required
	Amount of storage to free. (see Note 1).
ADDR	Required
	Address of the storage to free (see Note 2).
COMREGA	Optional
	(See Note 3)

Return Codes:

0	Storage freed successfully
4	Should not occur
8	Space to be freed was not allocated
12	Either the request is invalid or the FQE chain in the thread is corrupted

Notes:

1. "length" can be specified as a numeric constant, a numeric equate, as register content or a field containing the value.
   Examples:
   LEN=200 specifies a length of 200 Bytes
   LEN=(R3) length is contained in R3
   LEN=(*,LENGTH) length is contained in field LENGTH

2. "address" is the name of a field at the start of the area to be freed. As "length" It can also be specified in register or indirect field notation.
   Examples:
   ADDR=START free storage from label START
   ADDR=(R5) free storage addressed by R5
   ADDR=(*,STOR) free storage addressed in field STOR

3. For a Description of COMREGA refer to the section *How to use the Branch Entry Interface*.

The High Level Language Interface (HLLI)

Any programs using this interface to date will have fully compiled with the requirements for using branch entry and therefore require no changes to their source code. To use the branch entry interface, they must simply be linked with the stub module called TLOPUSER provided in the APS library. You should make sure that you always use the latest version of the module, which can be found in the latest APS cumulative fix pack loadlib.

Note that it is not possible to simply re-link a load module with TLOPUSER as the HLLI interface routines previously included must first be deleted using the linkage editor REPLACE statement. While simply linking in the new TLOPUSER will function correctly, the linkage editor will *NOT* delete references which were previously resolved using the older HLLI interface routines unless it is explicitly told to do so.

Programs which are not re-linked will continue to function normally, however, as the older HLLI routines used SVC entry to the nucleus, these routines will continue to experience the overhead of using this entry to the interface.

II Terminal Functions and Paging

This part of the Application Programming documentation covers functions related to terminals and terminal paging.

This information is organized under the following headings:

Terminal I/O Functions

Terminal Paging

5   **Terminal I/O Functions**

Concepts

During processing, an online application program is connected to a logical unit (LU). The LU can represent a terminal, either directly or indirectly, through another system (ACCESS, CICS Transaction Routing), or another application program (APPC sessions). The application program communicates with the LU by using Com-plete's *Terminal I/O* Functions. For asynchronous (non interactive) communication with terminals or network printers refer to the *Message Switching and Printout Spooling* and *Terminal Functions and Paging* sections in this documentation.

Communication with logical units is governed by conventions (protocols) that apply to each type of logical unit. The Terminal I/O Functions handle the corresponding protocols for the supported LU types in order to free the application program from controlling the sessions.

An application program communicates with an LU by using control commands contained in the data stream that control the processing and formatting of data. Two ways of handling terminal control are available for applications:

▪ Device-Dependent I/O

▪ Device-Independent I/O

Device-Dependent I/O

Device-Dependent I/O functions require the application program to to provide and manipulate all terminal control characters for both input and output operations within the data stream itself. These functions must be used*only* when a program is always executed from the same device type or when the program itself is able to recognize the device type from which it was started. Device-dependent I/O functions are *READS* and WRTS.

Device-Independent I/O

The Device-Independent I/O functions automatically supply the necessary control characters and are useful when an application program can be called from different LU types. Output formatting can be accomplished by inserting special characters in the output data stream. On input Com-plete removes all control characters inserted by the terminal before the data is passed to the application. The *READ, WRT* and *WRTT* functions provide device-independent I/O support.

Terminal Mapping

The functions above require data formats to be defined internally - in the application program. Every change in a field format or attribute requires modifications to the program. Input and output data formats can also be defined externally to the application program, in a separate module called*map*. The *READM* and *WRTM* functions use a map as basis of reference for field layout and attributes (terminal control characters). An application can optionally override the attributes defined in the map. Layouts and attributes can be changed without modifications to the program.

> **Notes:**

1. Device dependent and independent I/O functions work identically on LU6.2 sessions since the LU6.2 data stream does not contain control characters. Any character in the stream is treated as data.

2. Do not use Terminal Mapping functions in LU6.2 sessions.

3. Although output with option Done works similar to conversational, it is not possible to read data since the program is terminated without getting control.

4. Output with Return option followed by EOJ will terminate the program but the last record will not be readable by the operator the program is terminated just after displaying the message.

5. For LU6.2 sessions a the Done option works similar to Return + EOJ since there is no way to get a reply from the partner and the last record can be read normally at the other side.

6. The results of the carriage return character can be affected by the *CR* option in the *TIB* macro. Refer to section Defining Terminals and Printers in the *Com-plete System Programming* documentation for more information.

7. For each *New Line* placed consecutively in the output buffer, a shift for the next new line will be forced.

8. Delimiter lists cannot be used with the reread option

9. The Com-plete distribution source library contains sample delimiter lists for COBOL (COBDLST), PL/I (PL1DLST) and Assembler Language (CCDLM) that can be copied into the application program.

Programming Considerations

Program Logic

The Com-plete API makes handling of different protocols almost transparent to the application programmer. It automatically inforces most protocol rules in order to avoid runtime errors so the program design must take into account only the application requirements. It is important, however, to understand the internal processing logic and restrictions of the used functions in order to get the desired results:

■ Input functions do not cause any physical I/O to/from the partner LU; the data was already received either at program startup or as a result of a previous output function. The previously received data is just transfered from Com-plete's buffers to the application program's buffers. A *location counter,* updated for every input request, is maintained to determine how much data from the input buffer was already transferred to the application. The input buffer is freed only when the application received all input data.

- Output functions cause data/control information to be sent to the partner LU. The application is ROLLed OUT and gets control only after completion of the request. Conversational requests are put on the Ready To Run queue only after the reply data is received.

- Input requests may be issued only after conversational output requests, except for the first input after startup that returns the program name and initial data.

- Com-plete's buffer contents can be reread one or more times by specifying the *reread* option (suffix *R*). The reread option will prevent updates to the location counter, so the next request will transfer data starting at the same buffer location. Terminal dependent rereads must precede any terminal independent request because the latter will translate all remaining data to device-independent format.

Output Options

All Output requests (except READB) must specify a suffix that indicates the processing logic for the request. The suffix must be either:

R	Return.
	The application program's thread is ROLLed OUT, the data is sent to the terminal or partner LU but Com-plete keeps the right to send (no CD is sent). The application program is placed in the ready to run queue just after output completion. LU6.2 transaction programs remain in SEND State.
C	Conversational.
	The application program is ROLLed OUT, Com-plete sends the data and CD to the terminal or partner LU that now can send data back to Com-plete. When the reply is received the application program is put in the ready to run queue and can now issue a READ to retrieve the received data. LU6.2 application programs are now in RECEIVE state.
D	Done.
	This option works similar to a conversational output followed by EOJ. The difference is that the application does not get control - it is terminated normally after receipt of the operator reply. For LU6.2 sessions the conversation is terminated (CEB), the user logged off and the TIB is removed from the TIBTAB.

> **Notes:**

1. Although output with option Done works similar to conversational, it is not possible to read data since the program is terminated without getting control.

2. Output with Return option followed by EOJ will terminate the program but the last record will not be readable by the operator the program is terminated just after displaying the message.

3. For LU6.2 sessions a the Done option works similar to Return + EOJ since there is no way to get a reply from the partner and the last record can be read normally at the other side.

4. The results of the carriage return character can be affected by the*CR* option in the *TIB* macro. Refer to the section *Defining Terminals and Printers* in the Com-plete System Programming documentation for more information.

5. For each *New Line* placed consecutively in the output buffer, a shift for the next new line will be forced.

6. Delimiter lists cannot be used with the reread option

7. The Com-plete distribution source library contains sample delimiter lists for COBOL (COBDLST), PL/I (PL1DLST) and Assembler Language (CCDLM) that can be copied into the application program.

3270 Terminal I/O Handling

Device dependent I/O

When using device dependent I/O functions the application program must handle the whole data stream. All control characters the device places in the data stream are passed to the program on input. The program must also provide all control characters on output. Errors in control characters may cause unpredictable results.

Com-plete provides special facilities and handling for programmers using device-dependent I/O with IBM 327x compatible terminals.

For ease of use, all 3270 buffer addresses are referred to in the form of 2-Byte binary items relative to zero (16-bit addressing). Thus, row 1 column 1 is x'0000', row 2 collumn 3 is x'0052' (on 3270 model 2 terminals), etc... This holds true for both input and output. Com-plete translates all 12-bit row-column addresses into binary buffer offsets.

For *terminal dependent output* to 327x devices, the 1st output character is taken from the Write Cotrol Character (WCC). The WCC will not appear on the screen.

Only modified data fields from 3270 screens are read. On an initial input of a screen, the following data is presented to the application program:

```
                          Attention ID (AID) (1 byte)
                          Cursor address (2 bytes, zero-relative)
Repeats for     ⎧        Set-buffer-address (1 byte)
each modified   ⎨        Address of modified field (2 bytes, zero-relative)
field           ⎩        Data
```

Field data is variable in length so the most convenient way to process it (without using maps) is probably using a device dependent input (READS) with specifying X'11' (SBA) as delimiter. This enables the program to determine the exact data on a field by field basis. Refer to section *Delimiter Lists* for more details.

Device independent I/O

Application programs using device independent input will receive no control characters - all characters will be removed before data is transfered to the application. *Backspace* processing is performed for devices that transfer a backspace character (TTY devices).

For device independent output Com-plete supplies automatically all control characters required by the device in use (at runtime). Data is written/displayed beginning at the upper left-most margin of the page/left-hand corner of the screen using the maximum line length for the device. Outputs longer than 1 line continue on the next available lines. Further output formatting can be accomplished by using embedded special characters:

Character	Meaning	Action
X'00'	Null Character	No formatting action on most devices. Treated as space on some devices.
X'05'	Horizontal Tab	Translated into appropriate code for hardcopy terminal devices. Padding characters may be Required. Some devices ignore this code in the data stream.
X'0C'	Form Feed	Force a "Skip to Channel 1" condition. Ignored if the device does not support form feed.
X'0D'	Carriage Return	Carriage return without line feed.Treated as "new line" if the device does not support the carriage return character.
X'15'	New Line	Carriage return with line feed.
X'16'	Backspace Character	Overlays the previous character written. Accepted by most terminal devices.
X'25'	Line Feed	Causes a space down condition without carriage return.
X'40'	Space	Embedded spaces can be used to affect output formatting.

📄 **Notes:**

1. The results of the carriage return character can be affected by the*CR* option in the *TIB* macro. Refer to the section Defining Terminals and Printers in the *Com-plete System Programming* documentation for more information.

2. For each *New Line* placed consecutively in the output buffer, a shift for the next new line will be forced.

3. Delimiter lists cannot be used with the reread option

4. The Com-plete distribution source library contains sample delimiter lists for COBOL (COBDLST), PL/I (PL1DLST) and Assembler Language (CCDLM) that can be copied into the application program.

Delimiter Lists

An Application Program may optionally specify a delimiter list - one or more delimiter characters that are expected to be in the input data - instead of an amount of data to be read. Data is transferred up to (but not including) the delimiter.

The delimiter list specified in the DLIST argument is a working storage area in the application program. The format and contents of this area are illustrated in the following table:

Location	Length	Format	Contents
0	2	Binary	Number of Delimiters in this list.
0	2	Binary	Number of Characters returned. Must be initialized to zeros
4	2	Binary	Relative number of found delimiter in list.
6	2	Binary	Delimiters to be used

Notes:

1. Delimiter lists cannot be used with the reread option

2. The Com-plete distribution source library contains sample delimiter lists for COBOL (COBDLST), PL/I (PL1DLST) and Assembler Language (CCDLM) that can be copied into the application program.

Example

Assuming that the application sets the number of delimiters to 2, the delimiter list contains a comma and a period, and the data read by Com-plete that resides in the Com-plete buffer was:

```
JONES,JAMES ALFRED,719 HIGH ROAD.
```

the data returned to the application program by issuing a series of READ functions with the de-limiter list option specified would be:

Read Issued	Data Read	Number Returned	Delimiter Found
First read:	JONES	5	1
Second read:	JAMES ALFRED	12	1
Third read:	719 HIGH ROAD	13	2
Fourth read:	-	0	0

Terminal Mapping

Mapping transforms data by using a table called a map. A map defines both the output and input characteristics of a given terminal screen. The map is defined and stored external to the program as a separate load module. In the process of mapping, data fields in the application program and fields in the map are correlated with data fields on the terminal's screen. The map uses two types of fields, constant and variable. A "mapped terminal write" transfers constant fields from the map and variable fields from the application program to the terminal. A "mapped terminal read" transfers those variable fields that have been modified at the terminal to the application program. On output, the application program can override various map characteristics such as field atributes, cursor location, sound alarm, etc. Modifications to field attributes are done in reference to field names. Mapping optionally informs the application program, by field name, of the fields read and the fields incorrectly entered at the terminal.

A map can be created using the UMAP utility (see the Com-plete Utilities documentation) or assembled and linked using the mapping macros (see the section *Map Creation Using Macros*).

After the map has been created, the application program can perform terminal I/O using the map by specifying:

1. Mapping Request Control Blocks (MRCBs)

2. The location of the desired data within the application program's working storage area (optional)

3. The location of a Field Control Table (FCT) within the application program which contains field names and dynamic modification of field display characteristics (optional)

The application program consideration(s) for using a map and the process of creating a map are discussed in the remainder of this section. Further information on the use of the UMAP utility program can be found in the Com-plete Utilities documentation.

Map Contents

A map contains two types of data: global data and field data. Some of the information contained within the map may apply only to one type of 3270 (that is, the extended attributes of the 3279 graphics terminals are ignored for non-graphics 3270 terminals of the same size).

Global Data

Global information pertains to how the mapping system will deal with the map as a whole. Global data includes:

- The map name - used to verify that the correct map is being used

- The device code - used to signify the device type for which the map was designed

- Terminal Control Codes (TCCs) - used to specify control options to be used by the mapping system when using the map to write or read data such as Erase-the-Screen

- Global extended 3279 graphics attributes

The TCCs within a map can be overridden at execution time by specifying overriding TCCs within a field of the MRCB. The TCCs are defined in *Terminal Control Codes*. The extended graphics attributes can be overridden by field in the map or, at execution time, by field in the FCT.

Field Data

Field information pertains to how the mapping system will deal with a specific field. A field within a map can be defined as constant or variable.

A constant field is a field whose fixed text resides within the map. The application program cannot vary the text sent to the terminal or receive the text received from the terminal; however, the application program can modify the display characteristics of constant fields. Constant fields are used for displaying text whose contents are independent of the programs executing (that is, titles of a screen and field prompts).

A variable field is a field whose data resides within the application. Variable field data is moved by a mapped write (WRTMC, WRTM) call from the application storage to the terminal screen. If the field is defined as unprotected (that is, the data can be modified), the data entered on the screen can be returned to the application program with a mapped read (READM) call.

Both constant and variable fields share the following characteristics:

- An optional 6-byte field name;

- A screen location and field length;

- Field Description Codes (FDCs). The FDCs define the display and usage characteristics of a field. At execution time, the FDCs can be altered by specifying an override in the Field Control Table. FDCs describe characteristics such as high or normal display intensity, protected or unprotected. display or non-display, and required or Optional.
  Some FDCs are not valid for constant fields (e.g., unprotected, required, etc.). See *Field Descriptor Codes* for the definition of the FDCs;

- Color code for 3279 graphics terminals;

- Symbol set IDs for GDDM symbol set modules.

Variable fields have the following additional characteristics:

- Data buffer offset. The location of the data to be extracted for the write and the location for returning data from the terminal are determined by adding the data buffer offset to the address passed as the data buffer argument;

- Data types. Variable fields can be alphanumeric, zoned decimal, packed decimal, binary fullword, or binary halfword;

- Number of decimal places. The number of decimal places is used to display packed decimal, binary fullword, or binary halfword fields;

- Internal length. The internal length of packed decimal fields must be defined.

Map Names

The map is stored into the load library under a six-character name. The first four characters are the same as those specified in the name field of the UMAP menu or the MAPSTART macro statement. The last two characters are taken from the device code of the terminal for which the map was designed. UMAP displays this device code after the name field or it is taken from the DEVCODE argument of the MAPSTART macro.

For those applications/installations with performance problems due to excessive reloading of specific maps, the map can be placed into the resident list of Com-plete. Mapping will search first the thread, second the resident area of Com-plete, and finally the load library chain.

For maps accessed via the resident area of Com-plete that require scaling (see the following section), mapping will copy the map to the thread temporarily, scale the map, use the map, and free the thread storage.

Device-Specific Mapping and Scaled Mapping

This section describes the use of the device code and choice of using maps in a device-specific manner or in a scaled manner. **Terminal Device Type Codes** gives the device codes associated with each screen size of 3270 terminals.

Assume that your application uses only one screen. If this application program is required to operate from only one specific terminal device type, the 24-line 80-column F2, then one map is Required.
You could name this map XXXXF2 and set the MRCB as follows:

```
MAPNAME is 'XXXX'.
MAPVERS is 'B'.
  followed by three spaces.
MAPMVERS is space.                  (See Mrcb Exception Codes for MRCB format)
```

Mapping concatenates the XXXX to the device type, resulting in XXXXF2. If the application is executed from a different device type (for example, an F5), the application needs an XXXXF5 map

with the same field names and characteristics. These maps are device-specific; a unique map exists for each terminal type, as necessary.

In some situations, having a unique map for each device type allows the application to display more data on larger screens and less data on smaller screens.

Some applications take no advantage of the differences in 3270 screen sizes. If the screen layout of the map fits within the dimensions of another device type, the application can request mapping to use map XXXXF2 but scale the map to fit on the current device. To request scaling, set the MRCB as follows:

```
MAPNAME is six characters, i.e., 'XXXXF2'.
  followed by two spaces.
MAPMVERS is 'B'.
```

Note that scaling relocates the start of each field. Users of scaling should not use fields that wrap off of the right side of the screen and back on to the left.

Program Concepts

The application program *must* provide an MRCB in the working storage area of the application program. The MRCB contains the name of the map.

The program can optionally provide the FCT, if it is necessary to dynamically modify the display characteristics of specific fields or to receive more detailed information about input fields.

Since one map defines both the output and input handling, a typical application program performs a write-mapped call followed by a read-mapped call using the same map and same MRCB.

When a mapped read or write call is executed, mapping determines the name of the map by concatenating the MRCB map name field with the terminal device code. Mapping determines the location of the map and loads the map into thread storage, if necessary.

The manner in which individual fields are processed is determined by information passed in the MRCB and the optional buffer and FCT parameters. The MRCB is used to indicate that individual fields in the map are to have their processing characteristics controlled by the application program. When this indication is given, the CALL statement normally supplies the FCT parameters in which the overriding characteristics of the desired fields are specified.

The WRITE-OPTION of the MRCB allows the application program to indicate which of the following methods mapping should use:

■ Write all fields defined in the map, optionally overriding the display characteristics for those fields entered in the FCT;

■ Write only those fields specified in the FCT, optionally overriding the display characteristics.

The READ-OPTION of the MRCB allows the application program to choose among the following:

■ Read all readable fields;

■ Read only those fields specified in the FCT.

Note that a mapped write with no FCT and no buffer can be used to write only the constant fields.

The MRCB, FCT, and CALL statement conventions are discussed in detail in the following sections.

MRCB

The MRCB is a working storage area defined within the application program. It contains the name of a map, terminal control information, and mapping field control information. Note that an application program may contain more than one MRCB, but *only* one MRCB is Required.

Users are provided MRCB copy code for COBOL, PL/I, and Assembler. The format of the MRCB, along with a description of each of its fields, is found in *Mapping Request Control Block (MRCB)*.

In Assembler language, a map can be coded in line in the program and located immediately behind the MRCB. The program can then be assembled with the map located directly in line with the program, thus saving a load operation for the map. If this technique is to be used, the MAP-CONCAT fields in the MRCB must be initialized to a C.

Another method for specifying that the application has the map in storage is to set the MAP-CONCAT field in the MRCB to A and the MAP-ADDRESS field of the MRCB to the location of the map.

There is no logical restriction on how many maps a program can use. From a performance stand-point however, if multiple maps are to be used, it may be desirable to make some or all of them resident in the thread region of the application program. The MAP-COUNT field of the MRCB is used to request this option. This value literally creates a queue of thread resident maps. The number of entries in the queue is equal to the number specified in MAP-COUNT. If more maps are referenced than the queue can accommodate, the queue of maps is treated as a "first-in-first-out" queue. A map-count of zero signifies that the map should be used and then deleted.

Since the MRCB is used to pass control information back and forth between the application program and Com-plete, some of the MRCB fields must be set by the application program and some by Com-plete. Consequently, the MRCB is required for all mapping requests. The default value for all MRCB fields is spaces, with the exception of the MAPNAME and VERSION fields.

FCT

The FCT is defined within the application program in the working storage section. Its function is to enable the application program to change the display characteristics of individual fields and/or to receive more detailed information about each field.

The FCT, if defined, must consist of one FCT entry per field to be individually processed. Each entry is referred to as an FCTE.

If the FCT is not a parameter in the CALL statement, each field in the map is assumed eligible for writing, and all unprotected fields in the map are eligible for reading.

Each FCTE must be defined in one of three formats:

■ Short format of 6 bytes, or:

■ Long format of 10 bytes, or:

■ Extended format of 13 bytes.

Note that new applications must be coded using the extended format. The short and long forms have been retained only for compatability with existing programs.

The short form contains only field names. The long form contains the field name, an input flag, and a Field Descriptor Code (FDC) override field. The extended form also contains the override color and override symbol set ID for 3279 graphics. The format used must be indicated in the MRCB.

The format for each type of FCTE is defined in **Field Control Table (FCT)**.

Buffer Area

The buffer area(s), or record area, into which data is to be placed during a read operation and from which data is to be obtained during a write operation must be defined within the application program.

When an application program uses an existing file record definition, the programmer can specify the data offset during map creation. If an existing record format is not being used, use the UMAP edit copy code function to create a buffer.

Output Field

WRTM is the mapping function used for writing information or data to the terminal. This function is described in detail in the section entitled *Output Using Map: WRTM*.

Output processing involves global/field information from the map and dynamic overrides from the application program. Terminal Control Codes (TCCs) are stored in the map when the map is created, and they are overridden by specifying TCC-OVERRIDEs in the MRCB.

The TCC codes allow for the following sets of control:

E/N	E: Erase unprotected fields prior to the write. N: Specifies that these fields will not be erased. An application program can wish to rewrite only specific fields and have the remaining unprotected fields erased or not.
A/Q	A: Sound audible alarm at terminal. Q: Alarm is not sounded.
P/S	P: Start printer. S: Printer is not started.
K/M	K: Turn off the terminal's modified data tags. The modified data tags indicate that an unprotected field has been modified.
R/L	R: Unlock the keyboard. L: Lock the keyboard.

TCC codes affect the erasure and reformatting of the constant fields. In the following discussion, reference is made to options for which Com-plete determines whether an action is necessary. If these options are selected (B and F), Com-plete determines if the same application program and map was used for the previous write to the terminal and that no message switching, paging, terminal clear operation or program fetch was done. This determination should be sufficient to keep the screen correctly displayed with a minimum of rewriting of constant fields. Application programs that involve overlaying of mapped screens may need to force no erase or force the formatting of constant fields.

B/W	B specifies that Com-plete should determine whether the screen requires erasing before the write. W specifies that the screen should not be erased.
C/D/F	F specifies that Com-plete should determine if constant fields need to be rewritten. Specify D to force mapping to do no rewriting of the constant fields, or C to always force writing of the constant fields.

With these functions in mind, the application programmer can use the WRTM function to write the following information to the terminal:

▪ Write only those fields defined as constants in the map.

This option can be forced by not passing the buffer area or the FCT when executing the WRTM function.

▪ Write all the fields defined in the map, (optionally) overriding the display characteristics for those fields entered in the FCT.

This option can be forced by entering a space or an A in the WRITE-OPTION field of the MRCB, passing the buffer area in the WRTM function, and (optionally) passing the FCT argument.

- Write only those fields specified in the FCT, (optionally) overriding the display characteristics.

   This option is forced by entering an O in the WRITE-OPTION field of the MRCB and passing both the buffer area argument and the FCT argument.

Output validation is performed for data being written to the terminal to determine whether the program data area field contains data that can be properly edited and moved to the map field entry. Specific output validation performed is summarized below.

Alphanumeric and zoned decimal fields:

- Transferred from the program data area without validation.

Packed and binary fields:

- If the field contains invalid data, the program is terminated abnormally.
- Leading zeros are suppressed.
- If indicated in the map, a decimal point is edited into the display.
- A "-" immediately precedes either the high order digit or the decimal point, if the field is negative.
- For zero value fields, a single zero or the decimal point and all decimal places are displayed.
- A numeric attribute is forced, unless the field is specified as skip or protected (FDCs of S or P).
- If a value is too large to fit in the map display field, the display field is filled with asterisks.

Input Field Processing

READM is the mapping function used for processing input data from mapping requests. This function is described in detail in this section.

Input fields are processed according to location (that is, row and column) on the screen; therefore, the map used to read them should correspond exactly to the formatted screen. This can be easily accomplished by using the same map to both read and write the screen.

Input validation is automatically performed for data being read from the terminal to determine whether the input data can be properly edited and moved to the program field areas. The specific input validation performed is summarized below.

Alphanumeric fields:

- Validation is performed for length only.
- If more data is entered than the program-defined field can accommodate, an overflow exception condition will be posted.

- If not enough data is entered to fill the field, the data will be left-justified and space-filled.

Zoned decimal fields:

- Validation is for characters 0 - 9.

- Data can contain leading and/or trailing blanks.

- Data in the program data area will be right-justified and zero-filled.

- Possible exceptional conditions posted are INVALID DATA and OVERFLOW.

Packed fields:

- Input can have leading and/or trailing blanks.

- If negative, the first digit or the decimal point must be preceded by a "-".

- Decimal point placement is indicated by a period.

- Data must be numeric, except as indicated above. Data is aligned, converted, and stored in the program data field area.

- Possible input exceptions posted are INVALID DATA, OVERFLOW, and UNDERFLOW.

Binary fullword fields:

- Negative numbers are stored in two's complement form.

- Input can have leading and/or trailing blanks.

- If negative, the first digit or the decimal point must be preceded by a "-".

- Decimal point placement is indicated by a period.

- Data must be numeric, except as indicated above. Data is aligned, converted, and stored in the program data area.

- Possible input exceptions posted are INVALID DATA, OVERFLOW, and UNDERFLOW.

Binary halfword fields:

- Negative numbers are stored in two's complement form.

- Input can have leading or trailing blanks.

- If negative, the first digit or the decimal point must be preceded by a "-".

- Decimal point placement is indicated by a period.

- Data must be numeric, except as indicated above. Data is aligned, converted, and stored in the program data area.

- Possible input exceptions posted are INVALID DATA, OVERFLOW, and UNDERFLOW.

  The MRCB can contain a variable length feedback area. If so, this area is used to indicate input errors from the terminal. Data entered that conflicts with the field definition for the mapping field in which it is entered is not returned in the buffer area. Instead, the name of the mapping field, followed by an exception code, is listed in the feedback area. The MRCB feedback area exception codes are defined in **Mrcb Exception Codes**.

To illustrate the use of exception codes in the MRCB feedback area, consider the following example where the underscored data was entered at a terminal.

```
Gross Pay .01 SSN A00000000
```

If the field GPAY was defined as having less than three decimal places in the map, and if the field SSN was defined as a numeric field, then the MRCB feedback area would contain:

```
GPAY  UF,SSN  NN,
```

Note that each field entered in error is placed in the MRCB feedback area. The format of each entry in the feedback area is illustrated in the following figure. The number of fields in the feedback area is indicated by the MRCB's error-fields. Note also that the feedback area is not initialized between reads.

FIELD-NAME	OFFSET	LEN	Description
ERROR-FIELD	0	6	Name of the field in error
FILLER	6	1	Blanks
ERROR-CODE	7	2	Error codes, as described in **Mrcb Exception Codes**

In addition, three fields within the MRCB are updated. The MRCB's CURSOR-IN field contains the input field name cursor location. The FIELDS-READ and ERROR-FIELDS fields are used to indicate the number of fields returned to the application and the number of errors detected.

If an error is detected while processing a read function, a return code is posted in the return code field of the MRCB and in the*retcode*argument. The data for the field or fields in error is *not*transferred to the program data area.

In addition to the input exceptions posted in the MRCB feedback area, an input indicator is placed in each input flag field in the FCTE, if the long or extended format of the FCTE is being used (unless the FCTE is specified as ignored or protected). The codes returned are listed in *Field Control Table*.

Map Creation Using Macros

Before creating a map, you should design a separate display layout of each map for each terminal device type to be used. Currently, the only device types supported by mapping are 3277 models 1 and 2, 3278 models 1 through 5, and the graphics terminal or compatible devices. These device types are referred to as formattable devices; other device types are non-formattable devices.

After the map layouts have been designed, the macro statements defining the appropriate maps can be written. For example, consider the following display for which a map definition is desired:

```
NAME:    fred schwartz
ADDRESS: 1208 sw street
```

The map definition used to generate this display is:

```
MP01    MAPSTART F2
        MAPF     ,'NAME:'
NAME    MAPF     (1,10),14
        MAPF     (2,1),'ADDRESS:'
ADDR    MAPF     ,20
        APEND
        END
```

In the above sample map definition, note the entries accompanying the MAPSTART macro statement. The entry MP01 is the name of the map and the entry F2 is a terminal device type designator.

After the map has been defined, it is ready to be assembled and link edited. The assembly of the map should be performed using the ALGN option of the assembler or the results may be unpredictable.

MAPSTART Macro

The MAPSTART macro is used to identify (name) the map, specify the device class code of the terminal(s) on which the map is to be used, and specify optional terminal control information.

The format of the MAPSTART macro is:

```
name    MAPSTART  [devcode]
        [,FDCDEF=]
        [,TCC=]
        [,COLDEF=]
        [,PSDEF=]
        [,TYPEDEF=]
```

All the arguments, except *name*, are Optional.
These arguments are:

name	Required.Default: None.The name of the map. /The name must be exactly four or six alphanumeric characters in length and must begin with an alphabetic character.
devcode	OptionalDefault: F2 The device class code of the terminal for which this map is to be used.The devcode must be one of the following:
	F1 for 3270 model 1 or compatible device (12 lines x 40 columns).
	F2 for 3270 model 2 or compatible device (24 lines x 80 columns).
	F3 for 3278 model 3 or compatible device (32 lines x 80 columns).
	F4 for 3278 model 4 or compatible device (43 lines x 80 columns).

	F5	for 3278 model 1 or compatible device (12 lines x 80 columns).
	F6	for 3278 model 5 or compatible - (27 lines x 132 columns).
FDCDEF	Optional.Default: DTO; no extended attributes.The default to be used for the Field Descriptor Code (FDC) argument of the MAPF macro as used in this map definition.The available FDCs are listed in **Field Descriptor Codes**.	
TCC	Optional. Default: BEKQRSW The Terminal Control Codes (TCCs) to be used when performing write commands. See *Terminal Control Codes* for further details. If any TCCs are specified, at least one of the following pairs of TCC codes must also be specified: AQ, BW, DF, EN, KM, LR, or PS.	
COLDEF	Optional. Default: No default color assigned.The default color for the COL argument of the MAPF macros used in this map definition and the background color for the entire screen are: BL RE PI GR TU YE NE or blank, which applies to 3279 graphics terminals only.	
PSDEF	The default program symbol set ID for the PS argument of the MAPF macros used in this map definition.	
TYPEDEF	Optional. Default: AThe default for the TYPE argument of the MAPF macros used in this map definition:	
	A	Alphanumeric
	F	Fullword
	H	Halfword
	P	Packed
	Z	Zoned decimal

MAPF Macro

The MAPF macro is used to define each field to be displayed, including title fields and fields from the application program.

The specification of row and column locations for display fields must allow for a one-character filler entry that precedes each field in the display. For formattable devices, this field is reserved for the hardware-controlled attribute byte. For non-formattable devices, a blank is inserted in this location. This permits identical displays for both formattable and non-formattable devices.

The format of the MAPF macro is:

```
[name]    MAPF   [(row,column)]
               {,length1|,'constant'}
               [,length2]
               [,repeat]
               [,DECPLAC=]
               [,FDC=]
               [,COL=]
               [,PS=]
               [,OFFSET=]
               [,TYPE=]
               [,ITR=]
```

All the arguments are optional except *length1* and *constant*, between which a choice must be made. Note that the absence of the (row,column) argument must be shown by a comma. Fields must be specified in sequence, by column, within row, and can not overlap.

The arguments are:

name	Optional. Default: The field will be unnamed.The name is used in the FCT for field modification and feedback and in the MRCB feedback area during input field exception processing.The name must begin with a letter and cannot exceed six characters in length.
(row,column)	Optional. Absence of this argument must be specified by a comma.Default: The field is concatenated to the previously-defined field. If this is the first field defined, it is placed in location (1,1). The terminal display location for this field. The first terminal display position is (1,1). If (row,column) is omitted, the field immediately follows the previous field in this display. Note that an apparent space exists because of the attribute byte.
length1	Optional. Default: The length is derived from the length of constant, if entered; otherwise, it must be specified, or an error is generated.The display length for the field. For alphanumeric (type A) and zoned decimal (type Z) fields, it also specifies the data area field length within the application program using this map. This length does not include the filler byte.
constant	Optional. Default: If omitted, length1 must be specified.A character string to be placed in the display field. The display length of the field is determined by the number of characters entered in this argument. The maximum number of characters allowed is 255. The FDC for this field is forced to include S for format table devices.
length2	Optional. Default: Must be specified with packed fields.The data field length, as it exists in the application program for packed (type P) fields. The length is specified in bytes. The field cannot exceed eight bytes in length.
repeat	Optional. Default: 1The number of times, plus 1, that the constant is to appear in the terminal display in the same field. The length1 value for the field is derived by multiplying the length of the constant by the repeat factor.
DECPLAC	Optional. Default: 0 The number of decimal places in this field. This argument can only be specified if the TYPE is P, F, or H for this field. This argument cannot be specified if the constant argument was given.This argument is used to align the decimal point on input fields, and for editing decimal points on output fields. The maximum value is 15.

FDC	Optional. Default: DOTY The Field Descriptor Codes to be associated with this field. More than one FDC can be given. In case of conflict, the last one in the string takes precedence. If the constant argument was given, either S or P is assumed and cannot be overridden; however, any of the other allowable codes may be specified for the class of device for which the map is being used.
COL	The two-character color attribute to be associated with this field on a 3279 graphics terminal. The default is set from the COLDEF argument of the MAPSTART macro. Values: BL//RE//PI//GR//TU//NE or blank
PS	The one-character programmed symbol set ID to be associated with this field on a 3279 graphics terminal. The default is set from the PSDEF argument of the MAPSTART macro.
OFFSET	Optional. Default: The data field in the program working storage area is assumed to be concatenated to the last field specified with a positive offset, whether or not the offset was implied or specified.The number of bytes, either negative or positive, from the beginning of the buffer I/O area to the location of this field within the application program. The numerical value can range from -32768 to +32767. By adding this value to the data area argument passed in the READM or WRTM call, the location of the field in the program can be determined.
TYPE	Optional. Default: "A" or the value specified in the TYPEDEF argument of the MAPSTART macro.The type of field within the program data area.This argument must not be specified if the constant argument is given.
	<table><tr><td>A</td><td>Alphanumeric</td></tr><tr><td>F</td><td>Binary fullword</td></tr><tr><td>H</td><td>Binary halfword</td></tr><tr><td>P</td><td>Packed</td></tr><tr><td>Z</td><td>Zoned decimal</td></tr><tr><td>K</td><td>Kanji</td></tr></table>
ITR	Optional Default: OFFSpecifies whether input translation is to be performed on this field. This argument can only be specified for alphanumeric-type fields.

MAPEND Macro

The MAPEND macro allows for error detection, end-of-map processing, and the display of information about the map. The MAPEND macro is required.

The format of the MAPEND macro is:

```
MAPEND
```

Advanced Facilities

Structured Fields

Structured fields are used to convey additional control functions and data to or from the display terminal. Write Structured Fields (WSF) is the only 3270 command that can be used to send structured fields from the application to the display. This command may be used only for devices that support extended data stream. Devices that do not support WSF will reject this command with SENSE X'1003'.

Functions that can be accepted by display devices include partition/screen handling, outbound text or data streams and partition read. The display uses and AID (X'88') to indicate inbound structured field functions.

WSF commands can be issued using the *WRTSF* and *WRTSFC* functions. The application program must provide the complete command. *WRTSFC* followed by a device-dependent input function (*READS*) should be used if the command specifies an application-initiated read (Read Partition or Read Buffer).

For Read Buffer the application program can use the special function *READB* instead of *WRTSFC*. READB has no parameters and automatically sends the Read Buffer command.

For more information about WSF refer to the IBM 3270 Information Display System *Data Stream Programmer's Reference (#GA23-0059-1),* and the *3274 Control Unit Description and Programmer's Guide (#GA23-0061-2).*

Periodic Redisplay

Output functions cause the application program to be ROLLed OUT. A time parameter (Default=0) can be specified to tell Com-plete the elapsed time before the application program is put back in the ready-to-run queue. This feature can be used in *non-conversational* output requests to refresh a screen at periodic intervals either with updated data or a constant message. The *TESTAT* function (test for attention interrupt) can immediately follow the timed output function to terminate the output loop. Refer to the *Miscellaneous Functions and Function Tables* section for a description of *TESTAT*.

> **Note:** Application programs using this feature should not run on devices that do not support attention interrupts such as CICS Transaction Routing or LU6.2 sessions since there will be no way to interrupt the loop.

Time-out

For conversational output functions the application program is put in the ready-to-run queue only after the reply from the partner or device is received. If the *time* parameter is specified the application program, as in non-conversational functions, is put in the ready-to-run queue after the specified time is elapsed. If the application issues a subsequent input function before Com-plete received the reply no data will be returned to the application program, thus identifying a *time-out*. The application program can then take the necessary actions (backout, terminate, etc.). This feature is helpful to avoid program hangs on LU6.2 sessions or pseudo-terminals when the partner application fails to respond.

Syntax

Device-Independent Input: READ

```
READ[R] (retcode,area,length[,numleft|,numread[,dlist]])
```

Device-Dependent Input: READS

```
READS[R]  (retcode,area,length[,numleft|,numread[,dlist]])
```

Parameters:

R	OptionalReread Option.
retcode	Required.A fullword where Com-plete places the return code on completion of the operation.
area	Required. The buffer area in the working storage area of the application program where Com-plete places the data to be transferred from Com-plete's buffer.
length	Required.A binary halfword containing the number of characters to be transferred from Com-plete's buffer. length must be greater than zero.
numleft	Optional.A binary halfword where Com-plete places the number of characters remaining to be transferred before the READ without reread option was issued.
numread	Optional.A binary halfword where Com-plete places the number of data characters actually transferred from Com-plete's buffer to the application program buffer when a READ with reread option is specified.
dlist	Optional. Not applicable if the reread otion is used.The working storage area of the application program which contains the delimiter list to be used with the READ request. This area must have been previously defined and initialized by the application program.

Return Codes

Application programs should check the return code for one of the following values:

0	The amount of data transferred to the application program is equal to the amount of data in Com-plete's buffer.
4	The amount of data transferred to the application program is less than the amount of data in Com-plete's buffer.
8	The amount of data requested for transfer to the application is larger than the amount of data in Com-plete's buffer. Existing data is transferred, but extra buffer space is not modified.

Abends

An abnormal termination can occur during execution of the READ function. Possible causes include:

- An invalid argument was specified;

- The input area is not in the user area;

- The length specified is negative.

Input Using Map: READM

```
READM[R] (retcode,mrcb,darea [,fct])
```

Parameters:

R	Optional.Reread Option.
retcode	Required.A fullword where Com-plete places the return code upon completion of the operation.
mrcb	RequiredThe name of the MRCB as defined in the application program. The MRCB must be defined on a fullword boundary.
darea	Required.The name of the buffer data area in the application program where the input fields are to be placed.
fct	OptionalDefault: None. The name of the FCT in the application program that will be used according to the MRCB READ-OPTION field and the MRCB FCTE-FORMAT field and MRCB-FCTE count.

Return Codes

The return code, placed both in the first argument and in the RETURN-CODE field of the MRCB should always be examined for one of the following values:

0	No input errors were encountered and all required fields were read.
4	At least one error was encountered in the input. The MRCB ERROR-COUNT field contains the number of fields in error and the number of field names with exception codes in the feedback area.
8	There is at least one field in error and no feedback area was specified, or if a feedback area was specified, it is full.
12	The location of a field in the input does not match a field location specified in the map.

Data can be altered in the application program buffer area regardless of the return code value received.

Abends

An abnormal termination may occur during execution of the READM function. Some possible causes are:

- An invalid MRCB was found;

- An invalid area argument was specified;

- An invalid FCT argument was specified;

- The MRCB was not on a fullword boundary.

Device-Independent Output: WRT

```
WRT[T]{C|D|R} (retcode, area,length[,linelen[,time]])
```

WRT*T*x specifies a device-independent output function with*T*ext option. All data written to the terminal is separated into logical words that cannot be partially contained in one line. If the word does not fit completely on the line it is displayed on the next line.

Device-Dependent Output: WRTS

```
WRTS[E]{C|D|R} (retcode,area,length[,linelen[,time]])
```

WRTS*E*x specifies a device-dependent output operation with prior erasure of thet a 3270-screen.

Special Output

WRTSF - Write Structured Fields

```
WRTSF{C|D|R} (retcode,area,length[,time[,plist]])
```

READB - Write "Read Buffer"

```
READB
```

Parameters:

C/D/R	Required.Specifies the processing logic for the request. Refer to section Programming Considerations for more details.
retcode	Required.A fullword where Com-plete places the return code upon completion of the operation.
area	Required.A buffer area in the application program containing the data to be written to the terminal.
length	Required.A binary halfword containing the number of characters of data to be written.
linelen	Optional.A binary halfword containing the value of the logical line length to be used for the terminal. The linelen argument cannot be specified for terminal-dependent write requests. Default: If linelen is not specified, or if a linelen of zero is specified, the physical line length of the terminal is used.
time	Optional.A binary halfword containing the number of seconds after which the application program is placed at the bottom of the Com-plete ready-to-run queue to await dispatching. When used with the "return" form of the WRT request, control is returned to the application after the specified length of time has elapsed.When used with the "conversational" form of the WRT request, control is returned to the application when an interrupt occurs at the keyboard, or after the specified length of time has elapsed. The time-out can be identified by the fact that a "read" request returns no data.Default: None. The application program is placed immediately in the ready-to-run queue.

Return Codes

The application program must examine the first parameter after completion of the request for one of the following return code values:

0	The write operation was successful.
4	The write operation was terminated by the terminal user, either by pressing **CLEAR** (or its equivalent), or by entering the character string *EOJ. The application program can optionally choose to ignore this circumstance and continue normal execution.
8	The terminal operator has terminated the write operation by entering the character string *CANCEL, or the stack level has been terminated. If a terminal I/O with an option other than DONE is issued after a return code 8 is received, the application program is abnormally terminated. This value is normally reserved to enable the application program to perform logical end-of-job processing.

12	A terminal I/O error has occurred. When return code 12 is received, the application program is abnormally terminated, if another terminal I/O function that does not specify the DONE option is executed.
16	The output created by execution of the WRT function was destroyed at the terminal. Normally, this condition occurs if, while viewing the output, a message was sent to the terminal that destroyed the formatted output, or if the terminal user temporarily suspended this program in order to retrieve another. The application program should reissue the WRT request to force a rewrite of the entire screen. Mapping automatically handles this condition.

Abends

During execution of the WRT function, an abnormal termination may occur. Some possible causes are:

- Too many output lines were requested to be written.

- The area or length arguments were invalid.

- The terminal operator entered a reply of *CANCEL and the application program executed another WRT request other than WRTxD.

Output Using Map: WRTM

```
WRTM{C|D|R} (retcode,mrcb [,darea] [,fct])
```

Parameters

C/D/R	Required. Specifies the processing logic for the request. Refer to section Programming Considerations for more details.
retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
mrcb	Required.The name of the MRCB as defined in the application program. The MRCB must be defined on a fullword boundary.
darea	Required if the FCT parameter is specified. The name of the buffer data area in the application program where the output fields are obtained during WRTM processing.Default: If omitted, the format and fields from the map are written.
fct	Optional.The name of the FCT in the application program that is used according to the MRCB WRITE-OPTION field and the MRCB FCTE-FORMAT field. This argument need not be specified if all the fields specified in the map are to be written without modifying their display characteristics.

Return Codes

Return codes are placed both in the first parameter and in the MRCB RETURN-CODE field. Possible return codes and their meanings are:

0	Normal return.
4	The operator entered *EOJ or pressed **CLEAR**.
8	The operator entered *CANCEL in the first field, or the stack level has been terminated.
12	A terminal hardware error was detected during the WRTM operation.
16	The screen format has been erased. The screen format can be erased by the terminal that has received a priority message, or destroyed by a user who has suspended the application.

For formattable devices, it is assumed that the format is destroyed, if a return code other than zero is passed to the application program. In this situation, the format is automatically rewritten when the next WRTM request is executed, unless the WRITE-OPTION field of the MRCB specifies the letter O (only).

Abends

An abnormal termination may occur during execution of the WRTM function. Possible causes include:

- An invalid MRCB entry was given;

- An invalid area argument was specified;

- An invalid FCT argument was specified;

- The MRCB was not on a fullword boundary.

Extended Graphics Support

Extended 3279 graphics terminals have capabilities not supported by other 3270 models. Mapping support for these devices is implemented so that:

- Maps created for the non-extended will function on the extended models;

- The extended attributes will be ignored for non-extended models;

- Extended capabilities are defined on the basis of global and field definitions (no subfielding capabilities).

The extentions include color attributes, customized symbol sets, and extended highlighting.

These features are available with UMAP's TCC UPDATE function on the global level, and with UMAP's ATTRIBUTE UPDATE function on the field level. For example, you can specify the color

of the screen as pink, as well as a different color for each field. As before, these attributes can be overridden by use of the FCT.

Symbol Sets

Extended graphics terminals can be loaded with multiple user-defined symbol sets, which define the shape and color of any screen symbol. For further information on the creation of symbol sets, see the *IBM User's Guide for the Graphical Data Display Manager*.

The symbol sets are stored in either VSAM files or STEPLIB libraries as modules with eight-character names. Rather than specifying an eight-character name, mapping support refers to these symbol sets by a one-character symbol set ID. Applications must have the device loaded with the correct symbol set and symbol set ID.

Loading Symbol Sets

Symbol set modules can be loaded under application program control by using Com-plete's COLINK function and the Com-plete subroutine U2MASS. For the purposes of testing, the UMAP "LOAD PROGRAMMED SYMBOLS" function can be used to load symbol sets with an associated symbol set ID, and UMAP will call U2MASS.

Format

The format for using the COLINK function is:

```
COLINK (retcode,subroutine-name,entry1)
```

retcode	Required.A fullword where Com-plete places the return code upon completion of the operation.
sub-routine-name	An eight-character field with value "U2MASS".
entry1	An eleven-byte structure for each symbol set, to be:

entry1	
	<table><tr><th>Offset</th><th>Length</th><th>Contents</th></tr><tr><td>0</td><td>8</td><td>symbol set module name</td></tr><tr><td>8</td><td>1</td><td>symbol set ID</td></tr><tr><td>9</td><td>1</td><td>storage plane requested</td></tr><tr><td>10</td><td>1</td><td>storage plane assigned</td></tr></table>
	where:
	symbol set name Specifies the name of the GDDM-generated symbol set name module.
	symbol set ID Is a one-character ID by which mapping refers to the symbol set.
	storage plane

	Is a one-character storage plane name, requested as defined in 3270 component description, to be used by the module.
	storage plane Is assigned by U2MASS. used

Examples

Example 1 - Terminal-Independent I/O

This sample program demonstrates the use of terminal independent READ (also with reread option) and WRT using C, D and R options and also the time parameter. See the program comments for usage details.

```
COPY  CCGLOBS
*
*       REGISTERS ON ENTRY:
*               R2 = A(COMREG)
*               RD = A(CALLER'S SAVE AREA)
*               RE = RETURN ADDRESS
*               RF = ENTRY POINT
*
SAMP1    CSECT
         USING SAMP1,RC
         STM   RE,RC,12(RD)
         LR    RC,RF                    LOAD ENTRY POINT
         ST    RD,SAVE+4
         LR    R3,R1                    SAVE A(PARMS)
         LA    R1,SAVE
         ST    R1,8(RD)
         LR    RD,R1
         LA    R0,IPTAREA
         MVC   OUTAREA,BLANKS
         MVC   PROGNAME,BLANKS
         LA    R6,WSTABLE
         LA    R5,LASTENT
*
*       GET TOTAL LENGTH ENTERED
*
         CM$CALL READR,(RETCODE,IPTAREA,WRLEN,NUMREAD)
         LH    R1,NUMREAD
         CVD   R1,DWRD
         UNPK  TLEN(2),DWRD+6(2)
         OI    TLEN+1,X'F0'
*
*       SKIP PROGRAM NAME
*       *SAMP1 = 6 CHARACTERS + 1 BLANK = 7
```

```
*
         CM$CALL READ,(RETCODE,PROGNAME,PRGLEN)
         CLI   RETCODE+3,4            AMOUNT OF DATA TRANSFERED IS...
         BH    CANCEL                 LESS THAN REQUESTED. ERROR
         BL    DISPLY                 = REQUESTED ==> PROGNAME ONLY
*
*        THE TEST BELOW CAN BE USED INSTEAD OF THE PRECEDING:
*
*        CLI   NUMREAD+1,7           LENGTH ENTERED > 7?
*        BL    DISPLY                NO - ONLY PROGNAME
*
*        READ STARTUP DATA
*
CM$CALL READ,(RETCODE,IPTAREA,WRLEN,NUMLEFT)
         LA    R1,20
         CLI   NUMLEFT+1,20           RETCODE COULD BE TESTED INSTEAD        ↵

         BH    MOVE00
         LH    R1,NUMLEFT             ACTUAL LENGTH TRANSFERED
MOVE00   DS    0H
         BCTR  R1,0                   FOR EX
         EX    R1,MOVE1
         EX    R1,MOVE2               MOVE 1ST TABLE ENTRY
         LA    R6,20(,R6)             INCREMENT POINTER
*
*        DISPLAY STARTUP DATA
*
DISPLY   DS    0H
         LA    R1,L0
         STH   R1,WRLEN
         CM$CALL WRTC,(RETCODE,AREA0,WRLEN)
         ICM   RF,15,RETCODE
         BNZ   END                    NO, TERMINATE PROGRAM
*
WHOISIT  DS    0H
         MVC   IPTAREA,BLANKS         CLEAR INPUT AREA                       ↵

         CM$CALL READ,(RETCODE,IPTAREA,RDLEN,NUMLEFT)
         ICM   RF,3,NUMLEFT           ANY DATA?
         BZ    END                    NO
         CLC   IPTAREA(6),RECALL      ARE WE RECALLING TABLE ENTRY?
         BE    WHICHONE               YES, BRANCH
         MVC   0(20,R6),IPTAREA       MOVE INPUT TO TABLE
         LA    R6,20(,R6)             INCREMENT POINTER
WENTER   DS    0H
         LA    R1,L1
         STH   R1,WRLEN
*
*        WAIT FOR "TIME" SECONDS FOR OPERATOR REPLY
*
         CM$CALL WRTC,(RETCODE,AREA1,WRLEN,,TIME)
         ICM   RF,15,RETCODE          IF TIMEOUT TERMINATE
```

```
        BNZ    END                      NO, TERMINATE PROGRAM
        CR     R6,R5                    END REACHED?
        BNH    WHOISIT                  NO
        LA     R6,WSTABLE               RESTART FROM BEGIN
        B      WHOISIT                  READ NEXT
WHICHONE DS    0H
        LA     R8,WSTABLE               POINT TABLE START
        OC     IPTAREA+7(2),X2F0        BE SURE IT IS NUMERIC
        PACK   WRKCOUNT,IPTAREA+7(2)
        ZAP    TBLCOUNT,INCRP
        LA     R1,9                     LOOP COUNT
COMPARE CP     WRKCOUNT,TBLCOUNT
        BE     FOUND
        AP     TBLCOUNT,INCRP
        LA     R8,20(,R8)
        BCT    R1,COMPARE               TRY AGAIN
*
FOUND   MVC    NAME,0(R8)
        LA     R1,L2
        STH    R1,WRLEN
*
*       DISPLAY RECALLED ENTRY FOR "TIME " SECONDS.
*       NOTE THAT KEYBOARD REMAINS LOCKED
*
        CM$CALL WRTR,(RETCODE,AREA2,WRLEN,,TIME)
        ICM    RF,15,RETCODE
        BE     WENTER
*
END     DS     0H
        LA     R1,L3
        STH    R1,WRLEN
*
*       DISPLAY LAST MESSAGE AND TERMINATE THE PROGRAM
*
        CM$CALL WRTD,(RETCODE,AREA3,WRLEN)
*
CANCEL  DS     0H
        MCALL  ABEND,ABCODE=0001
*
*-------------------------------------------------------------
*       WORK
*-------------------------------------------------------------
MOVE1   MVC    OUTAREA(0),IPTAREA
MOVE2   MVC    0(0,R6),IPTAREA
DWRD    DS     D
SAVE    DS     18F
RETCODE DS     F
IPTAREA DS     CL80
NUMLEFT DS     H
NUMREAD DS     H
X2F0    DC     X'F0F0'
WRLEN   DC     H'80'
```

```
RDLEN     DC    H'20'
PRGLEN    DC    H'6'
INCRP     DC    PL2'1'
TBLCOUNT  DC    PL2'0'
WRKCOUNT  DC    PL2'0'
TIME      DC    H'10'
BLANKS    DC    CL20' '

RECALL    DC    CL6'RECALL'
AREA0     DS    0CL80
          DC    C'PROGRAM NAME= '
PROGNAME  DS    CL7
          DC    X'15'
          DC    C'TOTAL LENGTH ENTERED= '
TLEN      DS    CL2
          DC    X'15'
          DC    C'INITIAL DATA= '
OUTAREA   DS    CL20
AREA1     DS    0CL73
          DC    X'15'
          DC    X'15'
          DC    C'ENTER A 20 CHARACTER STRING OR ''RECALL'''
          DC    X'15'
DC    C'AND A 2 POSITION NUMBER FROM 1 - 10'
L0        EQU   *-AREA0
AREA2     DS    0CL80
          DC    X'15'
          DC    X'15'
NAME      DS    CL20
          DC    58C' '
L1        EQU   *-AREA1
L2        EQU   *-AREA2
AREA3     DC    C'PROGRAM TERMINATED NORMALLY'
L3        EQU   *-AREA3
          DS    0F
WSTABLE   DC    9CL20' '
LASTENT   DS    CL20' '
*
          LTORG ,
          COPY  CCREGS
          COPY  CCCOMREG
          END
```

Example 2 - Terminal-Independent I/O using delimiter list

This sample program does basically the same as the previous one, except that more than 1 table entry can be entered at each prompt, separated by commas.

```
COPY  CCGLOBS

*
*       REGISTERS ON ENTRY:
*           R2 = A(COMREG)
*           RD = A(CALLER'S SAVE AREA)
*           RE = RETURN ADDRESS
*           RF = ENTRY POINT
*
SAMP2   CSECT
        USING SAMP2,RC
        STM   RE,RC,12(RD)
        LR    RC,RF                   LOAD ENTRY POINT
        ST    RD,SAVE+4
        LR    R3,R1                   SAVE A(PARMS)
        LA    R1,SAVE
        ST    R1,8(RD)
        LR    RD,R1
        LA    R0,IPTAREA
        LA    R6,WSTABLE
        LA    R5,LASTENT
*
*       WRITE INITIAL MESSAGE
*
DISPLY  DS    0H
        LA    R1,L1
        STH   R1,WRLEN
        CM$CALL WRTC,(RETCODE,AREA1,WRLEN)
        ICM   RF,15,RETCODE
        BNZ   END                     NO, TERMINATE PROGRAM
*
AGAIN   DS    0H
        MVC   IPTAREA,BLANKS          CLEAR INPUT AREA
        CM$CALL READ,(RETCODE,IPTAREA,RDLEN,,DLMLIST)
        CLC   IPTAREA(6),RECALL       ARE WE RECALLING TABLE ENTRY?
        BE    WHICHONE                YES, BRANCH
        ICM   RF,3,DNUMRET            ANYTHING READ?
        BZ    DISPLY                  NO, END OF LIST
        MVC   0(20,R6),IPTAREA        MOVE INPUT TO TABLE
        LA    R6,20(,R6)              INCREMENT POINTER
        CR    R6,R5                   END REACHED?
        BNH   AGAIN                   NO
        LA    R6,WSTABLE              RESTART FROM BEGIN
        B     AGAIN
*
```

```
WHICHONE DS    0H
         LA    R8,WSTABLE              POINT TABLE START
         OC    IPTAREA+7(2),X2F0       BE SURE IT IS NUMERIC
         PACK  WRKCOUNT,IPTAREA+7(2)
         ZAP   TBLCOUNT,INCRP
         LA    R1,9                    LOOP COUNT
COMPARE  CP    WRKCOUNT,TBLCOUNT
         BE    FOUND
         AP    TBLCOUNT,INCRP
         LA    R8,20(,R8)
         BCT   R1,COMPARE              TRY AGAIN
*
FOUND    MVC   NAME,0(R8)
         LA    R1,L2
         STH   R1,WRLEN
         CM$CALL WRTC,(RETCODE,AREA2,WRLEN)
         ICM   RF,15,RETCODE
         BZ    DISPLY
*
END      DS    0H
         LA    R1,L3
         STH   R1,WRLEN
*
*        DISPLAY FINAL MESSAGE AND TERMINATE THE PROGRAM
*
         CM$CALL WRTD,(RETCODE,AREA3,WRLEN)
*
CANCEL   DS    0H
         MCALL ABEND,ABCODE=0001
*
*----------------------------------------------------------------
*        WORK
*----------------------------------------------------------------
SAVE     DS    18F
RETCODE  DS    F
IPTAREA  DS    CL160
NUMLEFT  DS    H
X2F0     DC    X'F0F0'
WRLEN    DC    H'80'
RDLEN    DC    H'20'
PRGLEN   DC    H'6'
INCRP    DC    PL2'1'
TBLCOUNT DC    PL2'0'
WRKCOUNT DC    PL2'0'
TIME     DC    H'10'
BLANKS   DC    CL160' '
RECALL   DC    CL6'RECALL'
AREA1    DS    0CL73
         DC    X'15'
         DC    X'15'
         DC    C'ENTER ONE OR MORE TABLE ENTRIES UP TO 20 CHARS LONG '
         DC    C'SEPARATED BY COMMAS OR'
```

```
            DC      X'15'
            DC      C'''RECALL'' AND A 2 POSITION NUMBER FROM 1 TO 10'

AREA2       DS      0CL80
            DC      X'15'
            DC      X'15'
NAME        DS      CL20
            DC      58C' '
L1          EQU     *-AREA1
L2          EQU     *-AREA2
AREA3       DC      C'PROGRAM TERMINATED NORMALLY'
L3          EQU     *-AREA3
            DS      0F
WSTABLE     DC      9CL20' '
LASTENT     DS      CL20' '
*
DLMLIST     DS      0F
DLMQUAN     DC      H'1'
DNUMRET     DC      H'0'
DLMNUM      DC      H'0'
DLIMITR     DC      C','
*
            LTORG ,
            COPY  CCREGS
            COPY  CCCOMREG
            END
```

Example 3 - Terminal-Dependent Output

This sample coding illustrates the use of a terminal-dependent output with erase option (*WRTSEC*). Note that Buffer addresses must be specified as binary values relative to zero (upper left corner).

```
SCREEN    START
  ...
  ...
  ...
WRTRTN    DS    0H
          CM$CALL WRTSEC,(RETCODE,SCREEN,IOLEN)
   OC     RETCODE,RETCODE
          BNZ   ERROR
  ...
  ...
ERROR     DS    0H
  ...
  ...
SCREEN DS    0D
  DC     X'C3'                   WCC
          DC      X'11'              Set Buffer Address
          DC      AL2(0)             Relatibe Buffer Address
          DC      X'13'              INSERT CURSOR
```

```
        DC    C'LINE 1'              DATA: LINE 1
        DC    X'11'                  SBA
        DC    AL2(80)                RBA
        DC    C'LINE 2'              DATA: LINE2
IOLEN   DC    H'20'
  ...
  ...
  END
```

Example 4 - Terminal I/O using Map

This example shows the use of macros to create a map for a defined screen layout and it's usage in terminal I/O functions.

Screen Layout

```
Name: .....name field.....
Addr: ....address field.......
City: ......city field........
SSN:  999999999       EMP.NR.: 9999.99
Gross Pay: 9999999


Enter Desired Action:  ..action..
```

Map definition

Below is an example showing how to define the map using macros. For ease of use, however, it is recommended to create maps using the UMAP Utility.

```
MAP1F2    START
MAP1F2    MAPSTART F2,TCC=KREBF,FDCDEF=R              alphanum., reqd
*
*   The field below is non-modifiable and outside the data area
*
ERRDIS    MAPF  (1,2),30,OFFSET=-40,FDC=S
          MAPF  (2,2),'Name:',FDC=SDKY                      CONSTANT
NAME      MAPF  ,20,OFFSET=0,A,FDC=UDKOY                    ALPHANUMERIC
          MAPF  (3,2),'Addr:',FDC=SDKY
ADDR      MAPF  ,24,OFFSET=20,A,FDC=UDKOY
          MAPF  (4,2),'City:',FDC=SDKY
CITY      MAPF  ,24,OFFSET=44,A,FDC=UDKOY
TSSN      MAPF  (5,2),'SSN:',FDC=SDKY
SSN       MAPF  ,9,OFFSET=68,TYPE=Z,FDC=UDKOY          ZONED DECIMAL
          MAPF  (5,25),'Emp.Nr.:',FDC=SDKY
EMPN      MAPF  (05,034),8,OFFSET=77,TYPE=F,FDC=UDKOY    BINARY FULLWORD
          MAPF  (6,2),'Gross Pay:',FDC=SDKY
GPAY      MAPF  ,7,4,OFFSET=85,TYPE=P,FDC=UDKOY,DECPLAC=2  PACKED 7,2
```

```
TENT     MAPF  (9,2),'Enter Desired Action:',FDC=SDKY
*
REQ      MAPF  ,10,OFFSET=-40,A,FDC=UDKOY
         MAPEND
         END   MAP1F2
```

> **Note:** The display characteristics of the constant fields labeled "TSSN" and "TENT" can be modified in the program since a name is specified.

Sample Program

```
SAMP4    CSECT
         USING SAMP4,RC
         ...
         ...
         MVC   MRCBNAM,=CL4'MAP1'     MAP MAP1XX
         MVI   MRCBVERS,C'B'          VERSION
         MVI   MRCBFCTF,C'L'          LONG FCT FORMAT
         MVC   MRCBFBAL,=H'30'        LENGTH OF FEEDBACK IS 30
         ...
INIT     DS    0H                     INITIALIZE DATA BUFFER
         MVI   ERROR,C' '
         MVC   ERROR+1(L'ERROR-1),ERROR
         MVC   NAME,ERROR
         MVC   ADDR,ERROR
         MVC   CITY,ERROR
         MVC   REQUEST,ERROR
         MVC   SSN,=CL9'000000000'
         MVC   GPAY,=PL4'0'
         MVC   EMPNUM,=F'0'
         ...
         ...
PROMPT   DS    0H                     RESET OPTIONS
         MVI   MRCBWOPT,C'A'          A-ALL FIELDS (SAME AS BLANK)
         MVI   MRCBROPT,C'A'          A-ALL FIELDS (SAME AS BLANK)
         CM$CALL WRTMC,(RETCODE,MRCB,DATABUFF)
         L     RF,RETCODE
         CH    RF,=H'16'              WAS SCREEN DESTROYED BY MSG?
         BE    PROMPT                 THEN REWRITE THE SCREEN
         LTR   RF,RF
         BNZ   EOJ
         CM$CALL READM,(RETCODE,MRCB,DATABUFF)
         L     RF,RETCODE
         LTR   RF,RF                  OK?
         BZ    GETCOMM                YES, GET COMMAND
         MVC   ERROR,MRCBFEED         SHOW ERROR
         B     PROMPT                 AND REWRITE SCREEN
*
GETCOMM  DS    0H                     GET COMMAND FROM OPERATOR
         XC    ERROR,ERROR            CLEAR MESSAGE
```

```
        LA    R0,COMFCTES              NUMBER OF FCTE'S
        STH   0,MRCBFCTC               SET COUNT
        MVI   MRCBWOPT,C'O'            LETTER O, SAYS O-NLY
        MVI   MRCBROPT,C'O'
        CM$CALL WRTMC,(RETCODE,MRCB,DATABUFF,COMMAND)
        L     RF,RETCODE
        CH    RF,=H'16'                WAS SCREEN DESTROYED BY MSG?
  BE    GETCOMM                  THEN REWRITE THE SCREEN
        LTR   RF,RF
        BNZ   EOJ
        CM$CALL READM,(RETCODE,MRCB,DATABUFF,FCT=COMMAND)
        L     RF,RETCODE
        LTR   RF,RF                    OK?
        BZ    PROCESS                  YES, PROCESS COMMAND
        MVC   ERROR,MRCBFEED           SHOW ERROR
        B     GETCOMM                  AND REWRITE SCREEN
*
PROCESS DS    0H
        ...
        ...
        MVC   ERROR,=CL30'RECORD SUCCESSFULLY UPDATED'
        ...
        ...
MESSAGE DS    0H               DISPLAY MESSAGE
        MVI   MRCBWOPT,C'A'           A-ALL FIELDS (SAME AS BLANK)
        MVI   MRCBROPT,C'A'           A-ALL FIELDS (SAME AS BLANK)
        MVC   MRCBCOUT,=CL6' '        BLANK CURSOR OUT FIELD
        LA    R0,SHOFCTES             NUMBER OF FCTE'S
        STH   R0,MRCBFCTC             SET COUNT
        CM$CALL WRTMC,(RETCODE,MRCB,DATABUFF,SHOONLY)
        L     RF,RETCODE
        CH    RF,=H'16'               WAS SCREEN DESTROYED BY MSG?
        BE    MESSAGE                 THEN REWRITE THE SCREEN
        LTR   RF,RF
        BNZ   EOJ
        B     INIT
*
        ...
        ...
EOJ     DS    0H
        CM$CALL EOJ
*
*-------------------------------------------------------------
*       WORK  AREA
*-------------------------------------------------------------
        ...
        COPY  CCMRCB                  FROM SOURCE LIB
COMMAND DS    0F
*             NNNNNN              FIELD NAME
*                  T              TAG
*                   FDC           OVERRIDING FDC
        DC    CL10'ENTER  D  '        CONSTANT   DISPLAY
```

```
          DC    CL10'ACTION UD '                    DISPLAY, UNPROTECT
COMFCTES EQU    *-COMMAND/10
*
SHOONLY  DS     0F
          DC    CL10'NAME   P'          NAME      PROTECT
          DC    CL10'ADDR   P'          ADDR      PROTECT
          DC    CL10'CITY   P'          CITY      PROTECT
          DC    CL10'SSN    P'          SSN       PROTECT
          DC    CL10'NUMBER P'          EMPLOYEE  PROTECT
          DC    CL10'GPAY   P'          GROSS PAY PROTECT
          DC    CL10'ENTER  N'          ENTER     NON-DISPLAY
          DC    CL10'ACTION PD'         ACTION    PROTECT, DISPLAY
SHOFCTES EQU    *-SHOONLY/10
*
DATABUFF DS     0F                      WRTMC, READM DATA BUFFER
ERROR    DC     CL30' '
REQUEST  DC     CL6' '
DRECORD  EQU    *                       RECORD BUFFER
EMPNUM   DC     F'0'
NAME     DC     CL20' '
ADDR     DC     CL24' '
CITY     DC     CL24' '
SSN      DC     ZL9'000000000'
GPAY     DC     PL4'0'
  ...
          ...
*
          LTORG ,
          COPY  CCREGS
          COPY  CCCOMREG
          END
```

Example 5 - LU6.2 TP

This example shows the use of I/O functions in LU6.2 sessions.

```
  COPY  CCGLOBS
*********************************************************************
*
*      SAMPLE PROGRAM FOR LU6.2 SESSION
*
*********************************************************************
*
*      REGISTERS ON ENTRY:
*            R2 = A(COMREG)
*            RD = A(CALLER'S SAVE AREA)
*            RE = RETURN ADDRESS
*            RF = ENTRY POINT
*
*********************************************************************
```

```
SAMP5    CSECT
         USING SAMP5,RC
         CMNAME BRANCH=OS
         STM   RE,RC,12(RD)
         LR    RC,RF                    LOAD ENTRY POINT
         ST    RD,SAVE+4
         LR    R3,R1                 SAVE A(PARMS)
         LA    R1,SAVE
         ST    R1,8(RD)
  LR     R3,R1                    SAVE A(PARMS)
         LA    R1,SAVE
         ST    R1,8(RD)
         LR    RD,R1
         MVC   PROGNAME,BLANKS
*
*        READ PROGRAM NAME
*        *SAMP5 = 6 CHARACTERS + 1 BLANK = 7
*
         CM$CALL READ,(RETCODE,PROGNAME,PRGLEN)
         CLI   RETCODE+3,4            AMOUNT OF DATA TRANSFERED IS...
         BNH   READ1                 LESS THAN REQUESTED. ERROR
         BAL   R9,CANCEL
*
*        READ 1ST RECORD
*
READ1    DS    0H
         CM$CALL READ,(RETCODE,RECORD1,RDLEN,NUMLEFT)
         OC    RETCODE,RETCODE
         BZ    READ2
         BAL   R9,CANCEL
*
*     READ 2ND RECORD FROM CHAIN
*     WRTC WITH LENGTH 0 MUST PRECEDE EACH READ
*

READ2    DS    0H
         CM$CALL WRTC,(RETCODE,RECORD1,ZEROLEN)
         OC    RETCODE,RETCODE
         CM$CALL READ,(RETCODE,RECORD2,RDLEN,NUMLEFT)
         OC    RETCODE,RETCODE
         BZ    READ3
         BAL   R9,CANCEL
*
READ3    DS    0H
         CM$CALL WRTC,(RETCODE,RECORD2,ZEROLEN)
         OC    RETCODE,RETCODE
         CM$CALL READ,(RETCODE,RECORD3,RDLEN,NUMLEFT)
         OC    RETCODE,RETCODE
         BZ    WRITE
         BAL   R9,CANCEL
*
*
```

```
*         NOW THE WHOLE CHAIN WAS READ AND WE ARE IN SEND STATE
*
WRITE    DS    0H
         CM$CALL WRTR,(RETCODE,PROGNAME,PRGLEN)
   CM$CALL WRTR,(RETCODE,RECORD2,WRLEN)
         CM$CALL WRTR,(RETCODE,RECORD1,WRLEN)
         CM$CALL WRTD,(RETCODE,RECORD3,WRLEN)
*
CANCEL   DS    0H
         MCALL ABEND,ABCODE=0001
*
*-------------------------------------------------------------
*         WORK
*-------------------------------------------------------------
SAVE     DS    18F
RETCODE  DS    F
PROGNAME DS    CL7
RECORD1  DS    CL25
RECORD2  DS    CL25
RECORD3  DS    CL25
BLANKS   DC    CL25' '
NUMLEFT  DS    H
NUMREAD  DS    H
ZEROLEN  DC    H'0'
WRLEN    DC    H'25'
RDLEN    DC    H'25'
PRGLEN   DC    H'7'
*
         LTORG ,
         COPY  CCREGS
         COPY  CCCOMREG
         END
```

A matching Client TP that may execute anywhere in the network (must run outside Com-plete) should contain an equivalent to the following LU6.2 verbs and options:

Verb	Options		
ALLOCATE	Remote Luname:	Com-plete APPL	
	MODEname:	installation-defined modename	
	TPname:	SAMP5	
	SECURITY:	valid Com-plete user ID and password	
	CONVTYPE:	MAPPED	
	SYNCLEVEL:	NONE or CONFIRM	
SEND	AREA:	25 Byte message	
	LENGTH:	25	
SEND	same as above		

Verb	Options
SEND	same as above
RECEIVE	will retrieve "*SAMP5 ", length=7
RECEIVE	will retrieve the 2nd message sent, length=25
RECEIVE	will retrieve the 1st message sent, length=25
RECEIVE	will retrieve the 3rd message sent, length=25 depending on the implementation also CEB (DEALLOCATE)
RECEIVE	will retrieve DEALLOCATE (if not retrieved in above verb)
DEALLOCATE	TYPE=LOCAL (depends on implementation)

6   Terminal Paging

Overview

The Com-plete terminal paging functions enable an application program to create a temporary disk file data set (referred to as a page file) and read or write data to the page file in blocks of data (referred to as pages). The pages of data in the page file can be dynamically displayed at the terminal from which the application program is executing without terminating the application program. The dynamic display of pages is accomplished by use of the Com-plete online utility program UP.

The page file is allocated as a temporary SD file data set, and is available for use only until another program is initiated from the same terminal. Each terminal can have only one page file per COMPASS stack level assigned to it at any time, and the page file assigned to one terminal cannot be accessed by an application program executing from another terminal. Each page file can contain a maximum of 255 pages.

A terminal user must use the Com-plete paging utility UP to display the contents of a page file. The use of the UP utility program is described in the *IBM User's Guide for the Graphical Data Display Manager* The UP functions used to display pages written to a page file are independent of the application program. The application program is not active while you use the UP paging utility to display data because the application program and the UP utility execute as separate subtasks of the operating system.

After using the UP utility functions to display the contents of a page file at the terminal, you can choose to change the contents of the page being displayed. Since the paging functions do not perform terminal I/O, the application program has to issue a terminal I/O READ function to obtain the information being displayed. The data is then written to the page file with a terminal paging WRITE function in order to force the change to the paging file.

The available paging functions are summarized in the following table. Each of these functions is described separately in this section.

Function	Description
POPEN	Allocates the SD file space required for the page file in use.
PWRT	Writes data to the page file.
PREAD	Reads data from the page file and places it in a working storage area in the application program.
PLIMIT	Restricts, or limits, the pages that can be displayed at a given terminal, based upon application-selected criteria.

POPEN Function

The POPEN function is used to allocate, or create, a page file. The page file created is assigned to the terminal from which the application program issuing the POPEN function is called. Since the POPEN function reserves an area for the page file, it must be the first paging function issued in the application program.

The POPEN function arguments enable the application program to determine the size of the page file. Page file size is determined by two items:

▪ The number of pages to be reserved for the page file

▪ The size of each page in the page file

Each item must be specified at the time the POPEN function is issued.

Format
>   The format for using the POPEN function is:

```
POPEN (retcode,length,numrec)
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
length	Required. A binary halfword containing the length of each page expressed as a number of characters.
numrec	Required. A binary halfword containing the total number of pages in the file. numrec can have a maximum value of 255.

Return Codes
>   A return code of 0 is issued upon normal completion of the POPEN function.

Abends
>   An abnormal termination may occur during execution of the POPEN function. Possible causes include:

>   ▪ An invalid *length* argument was specified;

>   ▪ An invalid *numrec* argument was specified;

>   ▪ Too large a page file was requested;

>   ▪ A disk error occurred during open processing;

>   ▪ Not enough space is available in the SD allocation pool to create the requested page file;

>   ▪ An incorrect number of arguments was specified.

PWRT Function

The PWRT function is used to write data to a page file that has been created with the POPEN function. The data to be written to the page file resides in a working storage area of the application program and is written on a page basis. The amount of data to be written cannot exceed the size of one page, but can be smaller than one page. Pages can be written either sequentially or directly.

Several PWRT options are available to assist in the formatting of pages. These format options include:

■ Erase the screen for a display terminal prior to displaying the page contents.

■ Insert a new-line symbol (X'15') into the page data to be displayed in order to format individual lines of output.

■ Treat the data in the page to be displayed as text, displaying the data in such a manner that words will not be split between lines.

■ Insert terminal-dependent control characters into the page data in order to format the page when displayed. In this situation, the application program should be executed from only one device type, since the control characters for one device type terminal may produce unpredictable results when used with another kind of device type terminal.

■ Do not automatically erase a display terminal screen when the page is displayed. This option might be used when a display type terminal is to be used and the application program is to create a page that, when displayed, will contain an explanation of a series of variables. The first page might be a list of all variables, and successive pages might then be used to explain each variable. Since the screen is not erased, the successive pages can literally add information to the screen.

Note that the size of a page in a page file can be less than or larger than the size of an output display for a specific terminal device type. If the terminal to be used is a display type terminal, then multiple pages can be displayed at one time.

PWRT causes a page to be written to the page file. If the terminal in use is a display type terminal, the screen is erased prior to a display operation when the page is requested for display.

PWRTT causes a page to be written to the page file. The data is treated as text and, when displayed at the terminal, words sre not split between two lines. If the terminal in use is a display type terminal, the screen is erased prior to a display operation when the page is requested for display.

PWRTS causes a page to be written to the page file. The data to be written to the page file must contain all terminal device-dependent control characters. These characters are used to format the terminal display when the terminal operator requests a display of the page. If the terminal in use is a display type terminal, the screen is not erased prior to a display operation when the page is requested for display.

PWRTSE causes a page to be written to the page file. The data to be written to the page file must contain all terminal device-dependent control characters. These characters are used to format the terminal display when the terminal operator requests a display of the page. If the terminal in use is a display type terminal, the screen will be erased prior to a display operation when the page is requested for display.

Format

The format for using the PWRT function is:

```
PWRT[T|S|SE] (retcode,area[,numrec][,length])
```

retcode	Required.A fullword where Com-plete places the return code upon completion of the operation.
area	Required.The buffer in the application program's working storage area from which the data is to be written.
numrec	Optional. A binary halfword that contains the number of the page to be written.Default: If not specified or zero, the page number is assumed to be one higher than the previously-written page. If a previous page has not been written, the page number is assumed to be first (page number 1).
length	Optional. A binary halfword that contains the length in bytes of the page to be written. The length must not be larger than the length of the page specified in the POPEN statement for the page file.Default: The page length specified in the POPEN statement for the page file.

Return Codes

A return code of 0 is issued upon normal completion of the PWRT function.

Abends

An abnormal termination may occur during execution of the PWRT function. Possible causes include:

- An invalid argument was specified;

- A function other than POPEN was specified, and POPEN was never specified;

- An incorrect number of arguments was specified;

- A disk error occurred while processing the page file.

PREAD Function

The PREAD function is used to read data from a page file. Data being read is placed in a working storage buffer area within the application program and is read on a page basis. The amount of data to be read cannot exceed a page in size, but can be less than a page. In addition, pages can be read either sequentially or at random.

After a page of data is read from the page file by the PREAD function, the application program can display the data to the terminal by use of the Com-plete terminal I/O functions. If you want to modify the contents of a displayed page, the application program must read the modified page from the terminal with a terminal I/O READ function and then write the page to the page file using the PWRT function.

Format

The format for using the PREAD function is:

```
PREAD (retcode,area [,numrec][,length])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
area	Required. The buffer area in the working storage area of the application program where the page data is to be placed.
numrec	Optional. A binary halfword containing the number of the page to be read.Default: Pages are read sequentially. If numrec is zero, the page to be read is assumed to be one higher than the previous page read. If a previous page has not been read, the first page is read.
length	Optional. A binary halfword containing the amount of data to be read from the page indicated. length must not be larger than the length specified in the POPEN function for the page file.Default: The page length specified with the POPEN function for the page file.

Return Codes

The following return codes are issued by the PREAD function.

0	Normal completion.
4	The page requested has not been written.

Abends

An abnormal termination may occur during execution of the PREAD function. Possible causes include:

■ An invalid argument was specified;

- The page file was not opened prior to issuing the PREAD function;

- An incorrect number of arguments was specified;

- A disk I/O error occurred while reading the page file.

PLIMIT Function

You normally display the pages of a page file using the Com-plete online utility program UP. This utility program is fully described in the Com-plete Utilities documentation. If you display pages from the page file, Com-plete keeps track of the following items:

- The number of the current page being displayed;

- The number of the highest page written to the page file by the application program.

For example, an application program has opened a page file with twenty pages using the POPEN function, but has only written ten pages to the page file using the PWRT function. Before any pages are displayed at the terminal, the current page and the highest page values within Com-plete will contain 1 and 10, respectively.

If you enter the command: `*UP/3`

to display page 3, the value for the current page is changed to 3, and the value for the highest page remains at 10.

If you enter the command: `*UP/N`

to display the next page, Com-plete determines the last page number displayed (in this case, page 3), increments the page count by one, and displays the next page (in this case, page 4). In this example, the current page value then becomes 4, and the last page value remains at 10.

If you enter the command: `*UP/P`

to display the previous page, Com-plete determines the last page displayed from the current page value (in this case, page 4), decrements the page count by one, and displays the page (in this case, page 3). In this example, the current page value then becomes 3, and the last page value remains at 10.

If you enter the command: `*UP/H`

to display the highest page written by the application program, Com-plete determines the value of the highest page (in this case, page 10), increments the page count accordingly, and displays the highest page written. In this example, the current page value then becomes 10, and the last page value remains at 10.

If you enter the command: `*UP/N`

after having displayed the highest page written by the application program, Com-plete displays an error message at the terminal, indicating that the application program did not create any more pages.

The PLIMIT function enables the application program to dynamically interrogate the page number of the current page and to change the value for both the current page and the highest page. This feature is accomplished by using the PLIMIT function to pass data back and forth to Com-plete using a working storage area in the application program.

The page number of the last page actually displayed on the terminal screen is optionally placed into the working storage area of the application program by using the PLIMIT function. In addition, the current page and the highest the page available for display is optionally set by the application program by use of the PLIMIT function. If the PLIMIT function is not used, Com-plete sets the value of the current page to 1 and the value of the highest page to the number of the highest page written to the page file by the program.

The PLIMIT function is useful in a variety of circumstances. For example, if you want to change the information contained in a particular page, the application program must use a Com-plete terminal I/O function to read your entry. The PLIMIT function can then be used to determine the appropriate page number of the page being modified before the PWRT function is used to actually write the change to the page file.

The PLIMIT function is also useful when there are several pages in a page file, not all of which are meaningful to the terminal operator. For example, if a paging file has 20 pages arranged as follows:

Pages 1 through 4 - instructions
Pages 5 through 13 - data
Pages 14 through 20 - programming notes

the PLIMIT function can be used to specify:

Current page = 5
Highest page = 13

If you enter the command: `*UP/C`

pages after page 13 are not displayed, allowing you to avoid viewing the information on pages 14 through 20. Since the initialized current page value is 5, a note could be written on the first logical page of data (page 5) indicating that instructions are contained on pages 1 through 4. You can then view pages 1 through 4 if desired (to read the instructions) by using a command such as: `*UP/1`.

Format

The format for using the PLIMIT function is:

```
PLIMIT (retcode,pagehi[,pagecur][,getpcur])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
pagehi	Required. A binary halfword containing the number of the highest page to be available for display. if pagehi is zero, the highest page actually written to the page file will be used.
pagecur	Optional. A binary halfword containing the number of the page to be displayed in response to the command *UP/C.Default: 1
getpcur	Optional. A binary halfword into which Com-plete places the number of the last page displayed by the terminal operator.Default: None.

Return Codes

A return code of 0 is issued upon normal completion of the PLIMIT function.

Abends

An abnormal termination may occur during execution of the PLIMIT function. Possible causes include:

- The page number requested is invalid;

- The page file was not open at the time the PLIMIT function was issued.

III   Storage Access Functions and Task Management

This part of the Application Programming documentation describes functions relating to access to external storage systems (including Adabas), as well as task managment functions.

This information is organized under the following headings:

Adabas & External Storage Access Functions

Task Management

7 Adabas & External Storage Access Functions

The external storage access functions enable an application program to read and write data to external storage devices. Five types of external storage access are available to Com-plete application programs:

- VSAM file I/O functions;

- ISAM and BDAM file I/O functions;

- SD functions;

- CAPTURE function;

- Adabas interface.

Application programs can read and write data with VSAM data sets using VSAM access methods. In an z/OS environment, application programs can use ISAM and BDAM data sets using Complete I/O functions. Note that data sets organized for access with an access method other than these cannot be accessed under Com-plete using standard Com-plete access functions.

SD functions enable the application program to read and write data either sequentially or directly to data sets defined and allocated by the Com-plete SD access method. SD data sets are normally logical data sets allocated within the Com-plete SD file. They can also be separate VSAM datasets (see *Com-plete Files and Associated User Files* in the *System Programming* documentation). The SD functions of Com-plete enable the application program to allocate, read and write, and delete the data sets.

The CAPTURE function enables the application program to capture, or write, data to the Complete Capture data set. Records can be written by the application program, but cannot be read. Information written to the Com-plete Capture data set can be selected, or read, from the Capture data set using the Com-plete batch utility program CUCTCAPT.

The Adabas interface enables your to access Adabas files. Adabas is the data base management system developed, marketed, and supported by Software AG.

All external storage access functions can be used in the same application program, providing a wide range of data storage and access techniques.

VSAM File I/O

The Virtual Storage Access Method (VSAM) can be used by Com-plete application programs. Full use of any type of VSAM data set or I/O function is provided; however, LOCATE processing of VSAM data sets is not supported by Com-plete, because the VSAM buffers are maintained in a storage key different from that of the application program.

Com-plete performs all VSAM requests without using an exit list. At the completion of the request, the user's ACB exits are called if there are any errors. Asynchronous journalling requests are not supported.

The use of VSAM under Com-plete is transparent to you and the application program. With the exception of the use of terminal I/O functions, a batch VSAM program will execute as a Com-plete online program with no modifications.

COBOL support of VSAM requires the use of a COBOL compiler (COBOL/VS) that supports VSAM file processing. In general, all COBOL facilities provided for VSAM support can be used.

PL/I support of VSAM requires the use of a PL/I compiler that supports VSAM file processing. In general, all PL/I facilities provided for VSAM support can be used.

In order to implement a program that uses VSAM processing techniques in a Com-plete environment, you must fully understand the interface requirements. The basic interface consideration can be summarized as:

- VSAM file definitions to Com-plete;

- VSAM file definitions in programs;

- VSAM file OPEN;

- VSAM file I/O;

- VSAM file CLOSE.

Each of these considerations is discussed below.

File Definitions to Com-plete

Application programs refer to files using DD/DLBL names. To establish the link between these names and the corresponding data sets, all DD/DLBL names referenced by application programs must be declared ("cataloged" to Com-plete using the subfunction FM of online utility UUTIL. This declaration includes the data set name, disposition (z/OS only), the name of the VSAM user catalog (z/VSE only), and other information (see below).

The data set is allocated to Com-plete dynamically when an OPEN request is issued against the appropriate DD/DLBL name and deallocated when Com-plete is stopped, or when the file is closed explicitly using the CLOSE or BATCH subfunctions of UUTIL FM.

In comparison with permanent data set allocation by JCL DD/DLBL statements, that was used with previous versions of Com-plete, this mechanism provides maximum flexibility of data set access by BATCH jobs and for data set maintenance (backup, restore, reallocation, rename, etc.) without the need of restarting Com-plete.

The file declaration in UUTIL FM not only contains the name of the data set, but also defines all parameters necessary for Com-plete to open the file. Options specified for a file in an application program are ignored. Instead, Com-plete uses the parameters defined via UUTIL FM to build control blocks and buffers once per DD/DLBL name. The parameters specified in the file definition to Com-plete must be consistent with the file processing techniques used by application programs.

All online programs referring to a given DD/DLBL name share the same control blocks and buffers. This allows efficient use of resources in the system, but also can significantly influence performance. Therefore, careful choice of parameter values is recommended for files referenced frequently and by a large number of terminal users.

The file definition parameters are described in the Com-plete Utilities documentation in the section *Function FM - File (DDN) Catalog Maintenance*.

File Definitions in Programs

File definitions in the application program should be created according to standard conventions established for the programming language being used.

From a Com-plete point of view, your only concern is that information placed in the application program must not conflict with the DDN catalog definition in the Com-plete program library.

File OPEN Statements

VSAM file OPEN statements in an online program are the same as those for a batch program. As in batch programs, the VSAM file must be opened via a standard OPEN statement before the first I/O request is made. In an online program, the OPEN statement is actually processed by Com-plete and not by z/OS/VS VSAM. Com-plete establishes the necessary linkage to enable the application program to access the VSAM data set. After a VSAM file is opened by an online program, the file remains open to Com-plete until Com-plete is reinitialized or until the file is closed explicitly using the FM function of the UUTIL utility. Subsequent OPEN processing is bypassed, except for establishing the logical connection between the application program and Com-plete. Com-plete uses the DDN cataloged file definition to actually open the data set. Thereafter, all accesses to the file by an application program uses the existing ACB control block structure. This processing is transparent to the application program.

Note that a VSAM file actually remains open to Com-plete, not the application program, after the termination of the application program. VSAM files are physically closed only at Com-plete termination time or by a status change in the file initiated by use of the FM function in the UUTIL utility program.

If Com-plete encounters an error in preparing the file for access, an appropriate VSAM FILE STATUS value is returned to the application program. The FILE STATUS is returned by Com-plete's OPEN processing or generated by Com-plete when indicating a non-VSAM error condition.

Non-VSAM FILE STATUS values that might be received from an OPEN request are:

136 x'88'	The file could not be opened because insufficient storage is available in the Com-plete region to build the necessary control blocks and buffers.Suggested response: Consult the system programmer responsible for Com-plete maintenance.
152 x'98'	Security errors. A password for the file was specified when the file was cataloged. This password was either not specified by the application program or, if provided, did not agree with the catalog entry. Suggested response: Provide the correct password or recatalog the DDN definition.
168 x'A8'	The DDNAME specified in the application program is invalid, not cataloged, or unavailable for online access because the file was placed in batch status. Suggested response: Ensure that the DDNAME specified in the application program is the same as that defined in UUTIL FM, and that online access to the file is enabled.

File I/O Operations

I/O operations to a VSAM file can proceed only if the file is successfully opened by the application program. The appropriate I/O request statements (START, READ, WRITE, DELETE, and REWRITE) can then be executed. Execution of these statements transfers control to Com-plete, and Com-plete performs the actual I/O.

Com-plete schedules the I/O request using VSAM, providing file integrity on a control interval level basis. If a record is requested that resides in a control interval being updated by another user, the request receives a VSAM FILE STATUS code of 96.

When Com-plete returns to the application program after processing an I/O request, the VSAM FILE STATUS value for the indicated file will have been set by Com-plete to either the value returned by VSAM or an appropriate value used to indicate a non-VSAM error detected by Com-plete.

Non-VSAM FILE STATUS values that might be received from I/O requests are:

24 x'18'	The file is no longer available. During processing of the application program and after the OPEN request has been processed, online access to the file was disallowed because another terminal user disabled access with the FM function in the UUTIL utility. Suggested response: Contact the system programmer responsible for Com-plete maintenance.
84 x'54'	LOCATE processing was requested. LOCATE processing is not supported by Com-plete.
153 x'99'	The SAF definitions for this User/File do not allow the access requested by the program (retrieve, update, add).
154 x'9A'	The type of processing requested (for example, update a record, add a record, read a record) conflicts with the options defined for the file in UUTIL FM.Suggested response: Correct the options defined in UUTIL FM.

File CLOSE Statements

When an online application program issues the CLOSE statement, Com-plete severs the logical connection between the program and the designated VSAM file(s). No subsequent I/O can be requested without requesting that the file(s) be reopened.

Com-plete automatically closes any VSAM file left open at termination of the application program; however, it is good practice and aids in system efficiency to close unused VSAM files. Note that the CLOSE statement causes a logical disconnection from the file and does not result in the actual close of the VSAM data set by Com-plete.

No FILE STATUS information is returned to the application program by Com-plete following a CLOSE operation.

ISAM & BDAM File I/O (z/OS Only)

ISAM and BDAM files are accessed using the Com-plete ISAM and BDAM interface functions summarized in the following table.

Function	Description
TFDEQ	Release an exclusively held file to enable access by other programs.
TFENQ	Place an exclusive hold on a file to exclude access by other programs.
TFGET	Retrieve one or more records from a file.
TFGETU	Retrieve a record from a file, with the intention of updating that record.
TFPUT	Add one or more records to a file.
TFPUTU	Write an updated record to a file as an update to that file.

Before any of these functions can be used successfully, the following conditions must exist:

- Each file or data set to be accessed must be defined and allocated in the Com-plete initialization procedure;

- Each file or data set to be accessed must have a DDN entry defined in the Com-plete file catalog. This procedure, referred to as "cataloging the DDN," is fully described in the Com-plete Utilities documentation in the section *Function FM - File (DDN) Catalog Maintenance*.

Existing batch programs that access BDAM or ISAM files may continue to access those files with no changes to their logic. In addition, batch programs accessing BDAM and ISAM files should use standard file access techniques, rather than Com-plete file I/O functions.

For BDAM files, only RECFM=F is supported. If records are blocked when the file is created, Com-plete performs automatic deblocking when the file is accessed. BDAM files must be allocated with DCB=OPTCD=R (this can be overridden on the DD statement for the file in the Com-plete startup

procedure). All BDAM modules should be placed in the pagefixed RAM/LPA list to avoid possible S606 abends.

For ISAM files, only fixed length records are supported. The maximum blocksize supported is 10K, and the relative key position must be larger than 0.

Retrieval of records for update is accomplished by using the TFGETU function. This function automatically forces an exclusive ENQ for the file containing the record to be updated. The exclusive ENQ is released when the TFPUTU function is executed to perform the actual record update. The ENQ/DEQ logic of the file I/O functions requires that the program be resident between execution of the two functions, TFGETU and TFPUTU; however, an ENQ remains in effect during a rollout caused by an Adabas call. No other terminal I/O functions or rollout functions can be used between execution of the TFGETU and TFPUTU functions.

If the update of a record in a file is dependent upon information in one or more records in other files, the other files *must* also have an exclusive ENQ placed upon them for the duration of the update process. This is accomplished by using the TFENQ function. After the update process is complete, the TFDEQ function is used to release the exclusive ENQ. As indicated for the TFGETU and TFPUTU functions, the use of the TFENQ and TFDEQ functions requires the application program to be resident for the duration of execution of these functions. Therefore, the application program can not issue a terminal I/O function or a ROLLOUT function between execution of the TFENQ and TFDEQ functions. Note that an ENQ remains in effect during a rollout caused by an Adabas call.

The use of multiple ENQs by more than one program should be considered carefully, since an interlock situation may occur. For example, if program A issues an ENQ for resource 1 and then resource 2, and program B issues an ENQ for resource 2 and then resource 1, a hard wait interlock situation will arise.

Request Parameter List

Use of the Com-plete file I/O functions requires definition of a working storage area within the application program called the Request Parameter List (RPL). The *RPL* contains information required by Com-plete to perform requested file I/O operations. A separate RPL can be defined for each file to be accessed, but only one RPL is Required.
The RPL may be shared by separate files, if the application program performs the necessary initialization functions; however, if more than one file is to be accessed simultaneously, separate RPL definitions must be established.

The format of the RPL is given in **Request Parameter List**.

TFDEQ Function (z/OS Only)

The TFDEQ function is used to release an exclusive ENQ placed on a file against which the TFENQ function was executed. The name of the file to be released is specified in the RPL. All other information in the RPL is ignored.

An ENQ caused by execution of the TFENQ function is effective only while the executing program remains in storage. If the active program is rolled out of storage by Com-plete before the TFDEQ function is executed, the ENQ is lost prematurely, and the TFDEQ function has no effect. An ENQ remains in effect during a rollout caused by an Adabas call, but it is cancelled automatically when a WRT or ROLOUT function is used. Therefore, if an application program issues a TFENQ function, neither terminal I/O functions nor the ROLOUT function can be executed until execution of the TFDEQ function is completed.

For example, if a program is performing a chained series of record additions to several files using data entered by a terminal operator, the program should read all the data from the terminal before issuing the TFENQ function against each file. After successfully adding the records to each file, the program should then issue the TFDEQ function for each file to release each exclusive ENQ.

Format

The format for using the TFDEQ function is:

```
TFDEQ (retcode,rpl)
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
rpl	Required.
	The name of the working storage area where the RPL is located.

The DDNAME field of the RPL must be initialized prior to issuing the TFDEQ function. All other RPL fields will be ignored.

Return Codes

The following return codes are issued by the TFDEQ function:

0	Normal completion.
4	A DEQ has already been issued for the file.

Abends

An abnormal termination may result during execution of the TFDEQ function. A possible cause is that the file specified by the DDNAME field of the RPL was not found.

TFENQ Function (z/OS Only)

The TFENQ function is used to force an exclusive ENQ on a file in order to guarantee file integrity while record information is being accessed. The name of the file to be ENQed is specified in the RPL. All other information in the RPL will be ignored.

When a record in a file is to be updated, the TFGETU function is used to retrieve the record, and the TFPUTU function is used to update the record. The execution of the TFGETU function places an exclusive ENQ upon the file and thus ensures file integrity for the specified update operation. The ENQ is subsequently released with the TFPUTU function; however, if the record to be updated is dependent upon record information in one or more additional files, no ENQ will have been established for the additional files.

The TFENQ function is provided to enable the application program to force an exclusive ENQ on one or more interrelated files. After the required record(s) have been processed, any outstanding ENQ for such files must be released using the TFDEQ function.

An ENQ caused by execution of the TFENQ function is effective only while the executing program remains in storage. If the active program is rolled out of storage by Com-plete before the TFDEQ function is executed, the ENQ is lost prematurely and the TFDEQ function has no effect. An ENQ remains in effect during a rollout caused by an Adabas call, but it is cancelled automatically when a WRT or ROLOUT function is used. Therefore, if an application program issues a TFENQ function, neither terminal I/O functions nor the ROLOUT function can be executed until execution of the TFDEQ function is completed.

For example, if a program is performing a chained series of record additions to several files using data entered by a terminal operator, the program should read all the data from the terminal before issuing the TFENQ function against each file. After successfully adding the records to each file, the program should then issue the TFDEQ function for each file to release each exclusive ENQ.

Format

The format for using the TFENQ function is:

```
TFENQ (retcode,rpl)
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
rpl	Required.
	Specifies the working storage item that contains the RPL for the indicated file.

The DDNAME field of the RPL must be initialized prior to issuing the TFENQ function. All other RPL fields will be ignored.

Return Codes

The following return codes are issued by the TFENQ function:

0	Normal completion.
4	An ENQ already exists for the indicated file, or the file is in batch status.
8	The application program has already ENQed the file.

Abends

An abnormal termination may occur during execution of the TFENQ function. A possible cause is that the file specified by the DDNAME field of the RPL was not found.

TFGET Function (z/OS Only)

The TFGET function is used to retrieve one or more records from a file. The file to be accessed is identified through the use of an RPL.

The DDNAME field in the RPL identifies the file to be accessed. When the read operation is performed against the indicated file, the number of records to be read is determined by the NUMBER-RECORDS field of the RPL. The number of records specified is read and placed in the working storage area of the application program. The recipient area is identified as an argument in the TFGET function.

The type of file being accessed, either BDAM or ISAM, is not indicated in the application program, but is known to Com-plete through the DDname definition in UUTIL FM; however, the use of the TFGET function arguments is dependent upon the type of access method to be used. Note that an improper combination of arguments will result in abnormal program termination.

Three positional arguments are provided:

- RPL identifier
- Buffer area identifier
- Record identifier

Files, whether BDAM or ISAM, can be accessed sequentially or directly. The choice of access is identified in the RPL. If a file is accessed sequentially, the third argument of the TFGET function is ignored, regardless of the file organization type. When a file is accessed directly, the third argument must be provided.

If a BDAM file is to be accessed directly, the record argument must be provided. The record(s) to be retrieved can be identified by a relative record number (relative to 1) or by its actual disk address (MBBCCHHR), specified in the record argument. If more than one record is to be retrieved, the records retrieved will be consecutive records beginning with the relative record indicated. If actual addressing (MBBCCHHR) is used, only one block can be retrieved.

If an ISAM file is to be accessed directly, the record argument must be provided. The record to be retrieved is identified by the key provided in the record argument. If more than one record is to be retrieved, the records retrieved are consecutive records beginning with the record whose key is specified in the record argument.

Format

The format for using the TFGET function is:

```
TFGET (retcode,rpl,area[,record])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
rpl	Required. The request parameter list for the file.
area	Required. The buffer area in the program where the record(s) will be placed.
record	Optional. Default: None. This field is ignored if the file being accessed is processed sequentially.A variable length field containing the information required to perform a direct retrieval of a record. The information in this field is used by both BDAM and ISAM access methods; the format of the field depends upon which access method is to be used. For ISAM files, this field contains the key of the record to be retrieved.

Return Codes

The following return codes are issued by the TFGET function:

0	Normal return.
4	An exclusive ENQ is outstanding for the file, or the file is in batch status.
8	An I/O error occurred while accessing the file.
12	An end-of-file condition has occurred.
16	A "no record found" condition has occurred. This return code will be received only if the record was accessed directly.

Abends

An abnormal termination may occur during processing of the TFGET function. Possible causes include:

- The RPL is invalid;

- The file identified by the DDNAME field of the RPL was not found;

- There is not enough free space in the application program thread region to accommodate the necessary I/O buffer.

TFGETU Function (z/OS Only)

The TFGETU function is used when updating a record in a file. When the TFGETU function is executed, an exclusive ENQ is directed against the file and the specified record is retrieved. The exclusive ENQ remains in effect until the record is rewritten using the TFPUTU function. No other program can access the file during this interval.

Note that an ENQ caused by execution of the TFGETU function is effective only while the executing program remains in storage. If the active program is rolled out of storage by Com-plete before the TFPUTU function is executed, the ENQ is lost prematurely. An ENQ remains in effect during a rollout caused by an Adabas call, but it is cancelled automatically when a WRT or ROLOUT function is used. Therefore, if an application program issues a TFGETU function, neither terminal I/O functions nor the ROLOUT function can be executed until execution of the TFPUTU function is completed.

The DDNAME field in the RPL identifies the file to be accessed. When the read operation is performed against the indicated file, *only* one record is retrieved. The NUMBER-RECORDS field of the RPL is ignored. The record specified is read and placed in the working storage area of the application program. The recipient area is identified as an argument in the TFGETU function.

The type of file being accessed, either BDAM or ISAM, is not indicated in the application program; it is known to Com-plete through the DDname definition in UUTIL FM. Appropriate use of the

TFGETU function arguments is dependent upon the type of access method to be used. An improper combination of arguments will result in abnormal program termination.

Three positional arguments are provided:

- RPL identifier
- Buffer area identifier
- Record identifier

Files, whether BDAM or ISAM, can be accessed sequentially or directly. The choice of access is identified in the RPL. If a file is accessed sequentially, the third argument of the TFGETU function is ignored, regardless of the file organization type. When a file is accessed directly, the third argument must be provided.

If a BDAM file is to be accessed directly, the record argument must be provided. The record(s) to be retrieved is identified by a relative record number (relative to 1) specified in the record argument.

If an ISAM file is to be accessed directly, the record argument must be provided. The record to be retrieved is identified by the key specified in the record argument.

Format

The format for using the TFGETU function is:

```
TFGETU (retcode,rpl,area[,record])
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
rpl	Required.
	The request parameter list for the file.
area	Required.
	The buffer area in the program where the record is to be placed.
record	Optional.
	Default: None. This field is ignored if the file being accessed is processed sequentially.A variable length field containing the information required to perform a direct retrieval of a record. The information in this field is used by both BDAM and ISAM access methods; the format of the field depends upon which access method is to be used. For ISAM files, this field contains the key of the record to be retrieved.

Return Codes

The following return codes are issued by the TFGETU function:

0	Normal return.
4	An exclusive ENQ is outstanding for the file, or the file is in batch status.
8	An I/O error occurred while accessing the file.
12	An end-of-file condition has occurred.
16	A "no record found" condition has occurred. This return code will be received only if the record was accessed directly.

Abends

An abnormal termination may occur during processing of the TFGETU function. Possible causes include:

▪ The RPL is invalid;

▪ The file identified by the DDNAME field of the RPL was not found;

▪ There is not enough free space in the application program thread region to accommodate the necessary I/O buffer.

TFPUT Function (z/OS Only)

The TFPUT function is used to add one or more records to a file. The file to be accessed is identified through the use of an RPL.

The DDNAME field in the RPL identifies the file to be accessed. When the write operation is performed against the indicated file, the number of records to be added is determined by the NUMBER-RECORDS field of the RPL. The number of records specified is read from the working storage area of the application program and added to the file. The RPL fields NUMBER-OPTIONS and SEARCH-OPTIONS is ignored during execution of the TFPUT function.

The type of file being accessed, either BDAM or ISAM, is not indicated in the application program; it is known to Com-plete through the DDname definition in UUTIL FM. Appropriate use of the TFPUT function arguments is dependent upon the type of access method used. Note that an improper combination of arguments will result in abnormal program termination.

Three positional arguments are provided:

▪ RPL identifier

▪ Buffer area identifier

▪ Record identifier

If a BDAM file is being accessed, the third argument, record, must be provided. The record to be added is identified by a relative record number (relative to 1) specified in the record argument. If more than one record is to be added, the records are added in sequential order, beginning with the relative record indicated.

If an ISAM file is to be accessed, the third argument must be omitted. The record to be added is identified by the key specified in the record itself. If more than one record is to be added, the records are added consecutively, based on their keys.

Format

The format for using the TFPUT function is:

```
TFPUT (retcode,rpl,area [,record])
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
rpl	Required.
	The request parameter list for the file.
area	Required.
	The buffer area in the program where the record(s) to be added is located. The NUMBER-RECORDS field in the RPL indicates how many records exist in this area.
record	Optional.
	Used for BDAM files only.Default: None.A binary fullword containing the relative record number of the first record to be added. If multiple records are to be added, the relative record numbers of the records to be added begin with the one specified in the record argument and are incremented by one for each succeeding record. If actual addressing (MBBCCHHR) is used, only one record can be written.

Return Codes

The following return codes are issued by the TFPUT function:

0	Normal completion.
4	An exclusive ENQ is outstanding for the file, or the file is in batch status.
8	An I/O error occurred while accessing the file.
12	A duplicate record was found while adding an ISAM record.
16	An add for a record was requested, but there was not enough space available in the file.

Abends

An abnormal termination may occur while processing the TFPUT function. Possible causes include:

- The RPL structure was invalid;

- The file identified by the DDNAME field of the RPL was not found;

- There was not enough free space available in the application program thread region to allocate the required I/O buffer;

- A relative record number was not specified when adding a record to a BDAM file.

TFPUTU Function (z/OS Only)

The TFPUTU function is used to rewrite one record to a file after it has been retrieved with the TFGETU function. The file to be accessed is identified through the use of an RPL.

The DDNAME field in the RPL identifies the file to be accessed. The record specified is read from the working storage area of the application program and rewritten to the file. The RPL fields NUMBER-RECORDS, NUMBER-OPTIONS and SEARCH-OPTIONS are ignored during execution of the TFPUTU function.

The type of file being accessed, either BDAM or ISAM, is not indicated in the application program; it is known to Com-plete through the DDname definition in UUTIL FM. Appropriate use of the TFPUTU function arguments is dependent upon the type of access method to be used. Note that an improper combination of arguments will result in abnormal program termination.

Two positional arguments are provided:

- RPL identifier;
- Buffer area identifier.

If a BDAM file is being accessed, the record to be rewritten is identified by the relative record number argument used in the last TFGETU function for the indicated file.

If an ISAM file is to be accessed, the record to be rewritten is identified by the key given in the record itself.

The TFGETU function is used to retrieve a record from a file when an update is to be performed to that record; however, the TFGETU function should always be preceded by the TFGET function. Any terminal I/O communication should be placed between the TFGET function and the TFGETU function. No terminal I/O must exist between the TFGETU function and the TFPUTU function.

When a record is retrieved, the information from that record is normally written to the terminal for display purposes with the intention of allowing the terminal operator to correct or modify the record for update purposes. The terminal write operation causes the application program to be rolled out of the thread, consequently nullifying any ENQ in effect for the designated file. Therefore, it is necessary to first retrieve a record with the TFGET function, perform the necessary terminal

I/O communication, obtain the record a second time with the TFGETU function, verify that it has not changed, and update the record with the TFPUTU function.

Format

The format for using the TFPUTU function is:

```
TFPUTU (retcode,rpl,area)
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
rpl	Required.
	The request parameter list for the file.
area	Required.
	The buffer area in the program where the record(s) to be updated will be located.

Return Codes

The following return codes are issued by the TFPUTU function:

0	Normal completion.
8	An I/O error occurred while accessing the file.

Abends

An abnormal termination may occur while processing the TFPUTU function. Possible causes are:

■ The RPL structure was invalid;

■ An update was requested for a record not retrieved with a TFGETU function.

SD Files

SD files are direct access files, or data sets, that are allocated, accessed, and maintained using the SD access method of Com-plete. Each SD file allocated is suballocated within the Com-plete system SD library COM.SD.

The common name SD acknowledges that SD files can be accessed either sequentially or directly. The Com-plete SD access method is available to application programs through use of the SD functions. The SD functions are summarized in the following table. Each of these functions is described in detail in a later section in this section.

Function	Description
SDOPEN	Create an SD file.
SDWRT	Write a record to an SD file.
SDREAD	Read a record from an SD file.
SDCLOS	Close an SD file.
SDDEL	Delete an SD file from the disk.

SD files contain fixed length records and can be processed either randomly or sequentially. The maximum size of an SD fileâ€ƒdepends on the installation parameters of the Com-plete SD library. The maximum number of SD files that can concurrently be in use by an application program is five.

The SD file functions provide a flexible access method that allows many different kinds of access. Any specific SD file can be simultaneously accessed or updated from applications executing from different terminals at different COM-PASS levels and from batch applications.

SD files are used in the following ways:

- As work files unique to an application session, (for example, editor work space);

- As online data collection files to be processed by batch;

- As a User Profile mechanism for tailoring applications to the preferences of specific users;

- As data distribution techniques from batch programs to online applications.

SD files are uniquely identified by the combination of NAME and TID parameters specified in the SD file function call. All SD file function calls for a specific SD file should have identical NAME and TID operands.

Note that any sense of shared versus exclusive use of an SD file is totally the responsibility of the conventions that you establish.

You must choose values for NAME and TID that are consistent with the NAME and TID values used by other applications. That is, some applications require unique disk work areas, but other applications need to share a work area. The SD files can manage both.

Shared SD Files

Applications that share an SD file use the same fixed NAME and TID operands. For example, if the message-of-today needs to be displayed by all terminals accessing an application, then the application could specify a NAME of "TODAY" and a TID of "SHR" in each SD file call. All users of the application would be using one SD file.

Note that for applications that write shared SD files, no SD read-for-update/update or write-next-unwritten record facility exists. If multiple programs are adding records to an SD file, it is advisable to keep the record number of the next free record in the first record of the SD file. Each application would then:

1. ENQ/LOCK on a serialization resource.

2. Read the first record.

3. Note the next free record number.

4. Increment the next free record number.

5. Write back the first record.

6. EQ/UNLOCK the serialization resource.

7. Write the noted record.

Unique SD Files

Applications that require a unique SD file by user must use a technique that guarantees a unique NAME and TID combination.

Usually the terminal user's user ID can be used as the NAME with a TID of "SHR" to ensure that an SD file is unique to a given user and that the SD file can be used by that user from any terminal.

If the SD file has meaning only during a single session with the application, then NAME could be specified as "APPL09" with the TID operand omitted. The TID could also be specified as the terminal ID, as retrieved by a GETCHR call. This allows the application to use the *hirec* operand on with SDOPEN or the length operand with SDREAD or SDWRIT.

A file name conflict may arise if a program that is simultaneously active on more than one level in a COM-PASS environment attempts to open an SD file. Com-plete provides a method to ensure that files (for example, work files) have unique names. The system does this by inserting the level number (a digit between 1 through 9) into the file name at user-specified positions. The required positions are denoted by high-value bytes (X`FF'). Note that the first character position of the file name cannot be modified in this way.

In addition, if the TID value specified is negative (that is, the high order bit is set), Com-plete will interpret this as a request for a level-dependent SD-file. The user can specify the level required in the bottom 4 bits if these bits are all set (that is, X'0F'), then Com-plete will use the current level.

An SD file can be read, written, or deleted simultaneously by two or more application programs. If two or more application programs are processing an SD file and one of them requests a deletion of the file, the file is physically deleted only after the last program accessing the file has issued a close or delete request for the file.

SDOPEN Function

The SDOPEN function is used to create a new SD file or prepare an existing SD file for access. The SDOPEN function must be used by an application program prior to accessing or deleting an SD file. More than one SD file can be opened simultaneously, as long as five or less SD files are being accessed by the application program. If more than five separate SD files are to be accessed, then at least one SD file currently open will have to be closed before issuing another SDOPEN function.

When the SDOPEN function is issued, the status of the SD file is indicated by the return code value. If the SD file is new, a return code of 0 is given. If the SD file already exists, a return code of 4 is given.

If a new SD file is being opened, the SD function arguments that must be specified are:

- File name;
- Record length;
- Number of records.

If an existing SD file is being opened, the file name argument must also be specified; however, the arguments that specify record length and number of records is ignored by Com-plete and used to return the existing record length and the existing number of records to the application program. Note that the record length and number of records cannot be changed for an existing file.

When an existing SD file is opened, the highest record written in the SD file can be identified with the SDOPEN function by having Com-plete return the record number of the highest record written. This feature can be used by an application program if it has a need to determine the last record written in the SD file at any one time. This feature can also be used to determine the last record to read when reading an entire SD file sequentially, thus avoiding reading records that have not been written.

If a program attempts to open an existing SD file, and Com-plete determines that the SD file does not exist, Com-plete assumes that the open request is for a *new* SD file; therefore, when opening an existing SD file, valid argument values should always be specified for record length and number of records. If, after checking the return code, the application program determines that a new SD file has been opened instead of an existing SD file, the SD file can be deleted and an error message written to the terminal.

Format

The format for using the SDOPEN function is:

```
SDOPEN (retcode,filnam[,length][,numrec][,tid] [,hirec])
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
filnam	Required.
	An eight-byte alphanumeric field that contains the name of the SD file being opened.
length	Optional.
	Required for a new SD file.Default: For an existing SD file, the record length will be returned. The maximum record length is 32752.For a new SD file, a binary halfword containing the length of each record in the SD file.
numrec	Optional.
	Required for a new SD file.Default: For an existing SD file, the number of records allocated to the SD file is returned.For a new SD file, a 2-byte binary field containing the number of records to be allocated for the SD file. The numrec for a new SD file remains as initialized in the application program. Since SD files are allocated on a track capacity basis, the application program can issue an SDCLOS function followed by an SDOPEN function for the SD file, and numrec will then contain the maximum number of records allocated for the SD file.
tid	Optional.
	Default: If the tid argument is omitted, the TID value for the terminal in use is assumed. The tid argument must be coded for an SD file being accessed by a batch program. For a new SD file, a binary halfword or a three-byte alphanumeric field containing the character string SHR. For an existing SD file, tid specifies a binary halfword. If the SD file was created with a tid value of SHR, then SHR must be specified.
hirec	Optional.
	Default: hirec is initialized to zeros by Com-plete for a new SD file. Com-plete returns the number of the highest record written for an existing SD file.For an existing SD file, a 2-byte binary field in which Com-plete returns the record number of the highest record written to the SD file.

Return Codes

The following return codes are issued by the SDOPEN function:

0	A new SD file has been opened.
4	An existing SD file has been opened.
8	An existing SD file has been opened, but the SD file is currently being used by another application program.
12	The SD file has already been opened by the application program.

Abends

An abnormal termination may occur during execution of the SDOPEN function. Possible causes include:

■ The SD file being opened is larger than the maximum size allowed;

■ The SD file directory is full;

■ An unrecoverable disk error has occurred.

SDWRT Function

The SDWRT function is used when writing records to an SD file. All records written are fixed-length records and can be written either sequentially or randomly.

The record to be written must reside in a working storage area of the application program. The application program can optionally write the entire record or a portion thereof.

When an SD file is created, each record is assigned a sequential number from one to the number of records requested. To write a record to an SD file randomly, the number of the record to be written must be specified. If no record number is specified, Com-plete writes the next sequential record to the SD file. Note that if this is the first SDWRT to a new SD file, *numrec* must be specified.

Note also that the length of each record to be written can be specified, but cannot be larger than the record length for the SD file as established when the file was created. If the length of data to be written is less than the record size for the file, the remaining characters of the record are padded with binary zeros. If the length of the record to be written is not specified in the SDWRT function, the record length established for the SD file during the SDOPEN function for the SD file is used.

Format

The format for using the SDWRT function is:

```
SDWRT (retcode,filnam,area[,numrec][,tid][,length])
```

retcode	A fullword where Com-plete places the return code upon completion of the operation.
filnam	Required.
	An eight-byte alphanumeric field that contains the name of the SD file to which the record is being written. The SD file specified by filnam must have been previously processed by the SDOPEN function.
area	Required.
	A buffer area in the working storage area of the application program that contains the record to be written.
numrec	Optional.

	Default: If numrec is not specified, the next sequential record is written. Required on the first SDWRT to a new SD file. A binary halfword that contains the number of the record to be written.
tid	Optional.
	Default: if tid is omitted, the TID value for the terminal in conversation is assumed. A tid must be specified for all batch programs.A binary halfword or a three-byte alphanumeric field that contains the character string SHR. The value must be the same as that specified when the file was opened.
length	Optional.
	Default: If not specified, the length is assumed to be the length specified in the SDOPEN function for the SD file.A binary halfword that contains the length of the data to be written.

Return Codes

A return code of 0 is issued upon normal completion of the SDWRT function.

Abends

An abnormal termination may occur during execution of the SDWRT function. Possible causes include:

- The SD file was not previously opened;

- The record number specified was greater than the maximum record number in the SD file;

- The record number specified was negative;

- An unrecoverable disk input/output error occurred.

SDREAD Function

The SDREAD function is used when reading records from an SD file. All records read are fixed-length records and can be read either sequentially or randomly.

The record to be read is placed in a working storage area of the application program. The application program can optionally read the entire record or a portion thereof.

When an SD file is created, each record is assigned a sequential number from one to the number of records requested. To read a record from an SD file randomly, the number of the record to be read must be specified. If no record number is specified, Com-plete reads the next sequential record from the SD file. If no records have been read yet, the first record is read.

Note that the length of each record to be read can be specified, but cannot be larger than the record length for the SD file as established when the file was created. If the length of data to be read is less than the record size for the SD file, the amount of data requested is read. If the length of the record to be read is not specified in the SDREAD function, the record length established for the SD file during the SDOPEN function for the SD file is used.

Format

The format for using the SDREAD function is:

```
SDREAD (retcode,filnam,area[,numrec][,tid][,length])
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
filnam	Required.
	An eight-byte alphanumeric field that contains the name of the SD file from which the record is being read. The SD file specified by filnam must have been previously processed by the SDOPEN function.
area	Required.
	A buffer area in the working storage area of the application program that contains the record after execution of the SDREAD function.
numrec	Optional.
	Default: if numrec is not specified, the next sequential record is read. If no records have been read, the first record is read. A binary halfword that contains the number of the record to be read.
tid	Optional.
	Default: If tid is omitted, the TID value for the terminal in conversation is assumed. A tid must be specified for all batch programs. A binary halfword or a three-byte alphanumeric field that contains the character string SHR. The tid value must be the same as that specified when the file was opened.
length	Optional.
	Default: If not specified, the length is assumed to be the length specified in the SDOPEN function for the SD file. A binary halfword that contains the length of the data to be read.

Return Codes

The following return codes are issued by the SDREAD function:

0	Normal completion.
4	The record requested has not been written to the SD file.
8	The record number requested is one larger than the last physical record in the SD file; this is the equivalent of an end-of-file condition.

Abends

An abnormal termination may occur during execution of the SDREAD function. Possible causes include:

- The SD file was not previously opened;

- The record number specified was more than one larger than the maximum record number in the SD file;

- The record number specified was negative;

- An unrecoverable disk input/output error occurred.

SDCLOS Function

The SDCLOS function is used to logically close an SD file. Because an application program can simultaneously process a maximum of only five SD files, the SDCLOS function must first be used if it is necessary to access more than five SD files.

Note that it is not necessary for an application program to issue an SDCLOS function. Com-plete closes the SD file when the application program terminates; however, it is recommended that SD files be closed when processing is completed. This facilitates more efficient operation of the system. In addition, since the SDOPEN function passes a return code value that indicates usage of an SD file by more than one application program, concurrent access of an SD file by more than one application program is logically communicated in proper access sequence.

Format

The format for using the SDCLOS function is:

```
SDCLOS (retcode,filnam[,tid])
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
filnam	Required.
	An eight-byte alphanumeric field containing the name of the SD file to be closed.
tid	Optional.
	Default: if not specified, the Terminal Identification number (TID) of the terminal in use is assumed. A tid must be specified by all batch programs that issue the SDCLOS function. A binary halfword or a three-byte alphanumeric field that contains the character string SHR.The tid must specify the same value as that specified when the SD file was opened.

Return Codes

The following return codes are issued by the SDCLOS function:

0	Normal completion.
4	SD still in use by another user.

Abends

An abnormal termination may occur during execution of the SDCLOS function. Possible causes include:

- The SD file was not opened;

- The *tid* argument was not specified in the batch program.

SDDEL Function

The SDDEL function is used to delete, or scratch, specific SD files. SD files to be deleted with the SDDEL function must be open at the time the SDDEL function is executed. Specifically, the SDCLOS function must not have been executed prior to executing the SDDEL function.

The SDDEL function identifies the SD file to be deleted by file name and TID. If an SD file was created with a TID value of SHR, then SHR must also be indicated in the SDDEL function.

If more than one application program is concurrently accessing an SD file and one of the accessing programs issues an SDDEL function for the SD file, the SD file is marked for deletion. However, the actual deletion occurs only after all application programs currently accessing the SD file have logically closed the SD file, either expressly by executing the SDCLOS function or implicitly by program termination.

Format

The format for using the SDDEL function is:

```
SDDEL (retcode,filnam[,tid])
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
filnam	Required.
	An eight-byte alphanumeric field containing the name of the SD file to be closed.
tid	Optional.
	Required for batch programs.

	Default: If not specified, the Terminal Identification number (TID) of the terminal in use is assumed. A binary halfword or a three-byte alphanumeric field that contains the character string SHR. If specified, the tid must have the same value specified as that when the SD file was opened.

Return Codes

The following return codes are issued by the SDDEL function:

0	Normal completion.
4	The SD file specified is currently being used by another application program; deletion is pending.

Abends

An abnormal termination may occur during execution of the SDDEL function. Possible causes include:

- The SD file was not opened;

- The *tid* argument was not specified by the batch program requesting the SDDEL function.

CAPTUR Function

The CAPTUR function is used to write data from a program area to the Com-plete capture data set. The data is written from an area specified by an area argument for the length specified by a length argument. An identification argument is provided that must be used to pass a six-character identification code to the CAPTUR function, if the program issuing the CAPTUR function is a batch program. The identification argument is optional in an online program.

When a record is written to the capture data set, Com-plete automatically adds a variable-length header that identifies the record being written. The format of this header is shown in **Captur Record Header**.

The CAPTUR function can be used to:

- Keep a record of information for use in reports;

- Store add or update transactions in an online environment and then perform the actual file add or update I/O operation in batch at a later time;

- Record audit information;

- Record diagnostic messages created by an application program. The messages can then be used to trace execution of a program and assist in finding errors.

Format
    The format for using the CAPTUR function is:

```
CAPTUR (retcode,area,length[,identifier])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
area	Required. The buffer area within the application program that contains the data to be captured.
length	Required. A binary halfword containing the length of the user data to be captured.
identifier	Optional. Default: If an identifier is not specified for an online program, the compressed name of the program issuing the CAPTUR function is placed in position zero of the CAPTUR record header. Required for batch programs. An alphanumeric field containing a six-byte identifier to be placed in position 14 of the CAPTUR record header. An identifier is required if the program issuing the CAPTUR function is a batch program, but is optional for online programs.

Return Codes

The following return codes are issued by the CAPTUR function:

0	CAPTUR was successful.
4	CAPTUR was not successful. This condition would arise if the CAPTUR feature were disabled.

Abends

Abnormal termination of the application program issuing the CAPTUR function can occur when the CAPTUR function itself abnormally terminates. Possible causes for this condition are:

- The *length* argument specified was invalid;

- An argument address is invalid;

- The identifier argument was omitted for a batch program.

Adabas Interface

Application programs that communicate with Adabas must use the standard Adabas calling sequence interface as illustrated below.

```
ADABAS (argument1,...,argument6)
```

The arguments available with the Adabas call statement are fully described in the *Adabas Command Reference* documentation.

Note that this call differs from other calls in that the RETCODE parameter must not be specified.

Multiple Adabas Nuclei

Access to multiple Adabas nuclei is accomplished by setting the file number in the Adabas Control Block (ACB).

The ACB file number is a binary halfword. Set the file number to the actual file number plus 256 times the data base ID.

If the data base ID is not specified, the default ID set at Com-plete initialization or as set by the installation's ULOPADAB exit.

> **Note:** Any ACB file number specified is cleared on return from Adabas.

The Com-plete/Adabas interface passes the supplied parameter list to a standard ADALNK module. Be aware that the ADALNK (and ULOPADAB) are entered in the A-mode of the calling routine: this means that if the program is running 31-bit mode, the parameter list must contain valid 31-bit addresses.

Return Codes Abends

The response to the Adabas call is returned in the Adabas control block. Please see the Adabas documentation for further details. Abnormal termination of the application program issuing the Adabas call may occur if the parameter list contains invalid addresses.

8   Task Management

The task management functions of Com-plete enable an application program to load, execute, or invoke another task or program.

In addition to these Com-plete functions, the application program can also choose to execute the following operating system functions:

▪ LOAD in z/OS, or CDLOAD in z/VSE;

▪ LINK in z/OS;

▪ XCTL in z/OS;

▪ DELETE in z/OS.

These functions are, by definition, not available to an application program written in COBOL or PL/I, except when used in a called Assembler subroutine.

Execution of the z/OS functions is functionally equivalent to that described for the Com-plete functions. The z/OS functions, however, will resolve a load request from the resident portion of the operating system, the resident portion of Com-plete, the thread region, or the COMPLIB library(s) of Com-plete in z/OS.

The following table lists the available task management functions:

Function	Description
ATTACH	Invoke another application program asynchronously.
CODEL	Delete a program that has been loaded into the application program area with the COLOAD function.
COEXIT	Return control to a user program.
COLINK	Pass control to a previously-loaded program. Control is returned to the program issuing the COLINK function.
COLOAD	Load another application program.
COXCTL	Transfer control to a previously-loaded program. Control is not returned to the program issuing the COXCTL function.
FETCH	Fetch a program from the Com-plete program library, pass control to the fetched program, and (optionally) pass data to the fetched program.
LOAD	Load and initialize a table or module into the application program area and (optionally) pass control to the table or module.
SCHED	Allow the scheduling of a user task or transaction.

ATTACH Function

The ATTACH function causes a specified program to be loaded into a thread and executed asyncronously with the calling program. The program to be attached can reside:

- in the resident area of the operating system, or:
- in the resident area of Com-plete, or:
- in the application program thread region area, or:
- in the Com-plete COMPLIB load library chain.

The thread selected for execution of the attached program is determined by the attributes of the attached program, *not* by the calling program.

Data can be passed to the attached program using the *area* and *length* arguments of the ATTACH function. The attached program can retrieve this data by issuing a terminal READ function. This read must be the first terminal I/O operation issued by the attached programs; otherwise, the data transferred from the calling program is lost.

Attached programs have some limitations on the types of functions they can perform. These limitations are:

1. All terminal write functions performed by an attached program are converted to class 1 messages. The destination terminal is the terminal from which the calling program is executing.

2. The attached program cannot perform a conversational write (WRTC) function. If a WRTC function is issued, it is treated as a WRTD function, and the attached program terminates.

3. All terminal write functions performed by an attached program must be device-independent. Attempts to perform a device-dependent write causes the attached program to be terminated abnormally. Terminal mapping is a device-dependent function and, as such, causes abnormal termination of the attached program.

4. Error messages issued by Com-plete for the attached program will be sent to the terminal from which the calling program is executing via a class 1 message. If the attached program terminates abnormally, an online dump is taken in the normal manner.

Several programs used in an online environment are suited for attached applications. These programs characteristically perform functions that are relatively long and do not require completion prior to continuation of input. Examples are programs that printout spool large reports or generate a batch of updates for a file.

Since attached programs execute asynchronously with the calling program, the terminal in use by the calling program is available to perform other functions while the attached program functions are being performed. Attached programs are assigned a dummy terminal while executing, and the programs are assigned the lowest priority for thread scheduling. It is important that attached

programs be designed not to remain in a thread for long periods of time without performing a terminal WRITE function or a ROLOUT function. The execution of a terminal WRITE or a ROLOUT function permits higher priority programs to be scheduled for execution in the thread.

Format

The format for using the ATTACH function is:

```
ATTACH (retcode,name[,area,length][,userID])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
name	Required. An eight-byte alphanumeric field containing the name of the program to be attached. The name must be left-justified and padded with blanks.
area	Optional, no default.The name of the buffer area that contains data to pass to the attached program. The attached program must issue a READ function in order to retrieve this data. If this argument is specified, length must also be specified.
length	Optional, no default.A binary halfword containing the length of the data to be passed to the attached program. Maximum length is 4094 bytes.
UserID	Optional, no default.An eight-byte alphanumeric field containing the user ID of the allocated program. It is left-justified and padded with blanks. If an external security system is active (RACF, ACF2 or TopSecret), Com-plete verifies that the user ID under whose control the ATTACH function executes is authorized in the SURROGAT class to submit jobs for the user ID specified in this field.

Return Codes

The following return codes are issued by the ATTACH function:

0	No errors.
4	Unable to perform the ATTACH function because no dummy TID is available for use by the attached program. If this problem happens frequently, ask the system programmer responsible for Com-plete maintenance to increase the value of the NOTIBS keyword argument in the TIBTAB definition.
8	The program to be attached has not been found.
12	Security error. The user running the program is not authorized to access the requested program.
16	The name of the program to be attached is not valid.
20	No space available in general buffer pool.
24	A user ID is specified, but either this user ID is not defined, or the current user is not authorized to start programs with this user ID.

Abends

An abend may occur during the ATTACH operation. Possible errors include:

- Invalid *name* argument;

- Invalid *area* or *length* argument;

■ An attempt was made to attach a planned overlay type program.

CODEL Function

The CODEL function causes a program that has been loaded into the application program area to be physically deleted from the application program area. The deletion will occur only if the CODEL function has been issued as many times as the COLOAD function has been executed.

The CODEL function is used only after the COLOAD function. The COLOAD function is used to load programs into the application program area. Each time a COLOAD function is successfully executed, a use count of loads is maintained and incremented. When a CODEL function is executed for the same program name, the use count is decremented. If the use count becomes zero and the copy of the program is not currently being used to satisfy either the COLINK or COXCTL functions, the copy is physically deleted from the program area. If the copy is currently being used to satisfy a COLINK or COXCTL function, the copy is deleted only when it is no longer needed by these functions, and only if the use count is still zero. This use count can become negative if more than 32,767 loads are issued with no intervening deletes. In this situation, the loaded program cannot be deleted.

A program that has been loaded via the COLOAD function and not deleted via the CODEL will occupy storage until the calling program terminates.

If the CODEL function is issued for a module that has been loaded via an SVC LOAD (that is, a supervisor call, whether it is an Assembler subroutine or a compiler-generated call), the effect is the same as if an SVC DELETE had been executed.

Format
   The format for using the CODEL function is:

```
CODEL (retcode,name)
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
name	Required. An eight-byte alphanumeric field containing the name of the program whose use count is to be decremented. The name must be left-justified and padded with blanks.

The use count for the program identified by name is decremented by one. If the use count becomes zero, the storage occupied by the program becomes available for other use immediately, or as soon as it is no longer required for COLINK or COXCTL requests.

Return Codes
   The following return codes are issued by the CODEL function:

0	No errors. The use count has become zero, and the program has been deleted unless it is currently being used by a COLINK or COXCTL request.
4	The program was not deleted because the use count was not zero or the program has been loaded more than 32,767 times more than it has been deleted. In this situation, the use count becomes negative, and the program cannot be deleted.

Abends

An abnormal termination may occur during execution of a CODEL function. Check to see whether the *name* argument is invalid.

COEXIT Function

The COEXIT function may be used to return control to a user program which issued a COLINK to the current program, or where the current program was entered via an XCTL request, to the program that linked to this program. If control was given directly from Com-plete, issuing this function will terminate the application and return control to Com-plete.

The format for the COEXIT function is:

```
COEXIT
```

There are no parameters associated with the COEXIT function.

Abends

There are no abends normally associated with the use of the COEXIT function.

COLINK Function

The COLINK function causes control to be passed to a specified program. The program to receive control can reside:

■ in the resident area of the operating system, or:

■ in the resident area of Com-plete, or:

■ in the application program thread region area, or:

■ in the Com-plete COMPLIB load library chain.

Note that programs that use the z/OS planned overlay concept cannot be loaded using the COLINK function. The program indicated by the COLINK function returns control to the calling program in one of the following situations:

- COBOL - GOBACK;

- PL/I - RETURN;

- Assembler - BR R14.

If the COLINK program does not reside in memory and it has not previously been loaded using a COLOAD or LOAD SVC function, Com-plete will load the program indicated from the Com-plete COMPLIB load library chain into the application program thread region area prior to passing control. The COLINK function, without an accompanying COLOAD function, should be used for modules requiring one-time processing. The storage occupied by the colinked program will then be temporary storage only, and will be freed when control is returned to the program that executes COLINK.

The COLINK program is deleted from the application program area upon return to the calling program except when it has previously been the object of a COLOAD function and not that of a subsequent CODEL function.

If a program has been loaded via either the COLOAD function or an SVC LOAD, the loaded copy of the program is used to satisfy any load type request (that is, COLOAD, COLINK, COXCTL, CODEL, SVC LOAD, SVC LINK, SVC XCTL, SVC DELETE).

Format

The format for using the COLINK function is:

```
COLINK (retcode,name[,arg1]...[,argn])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
name	Required. An eight-byte alphanumeric field containing the name of the program to which control is to be given. The name must be left-justified and padded with blanks.
argn	Optional. Any parameter(s) to be passed to the program identified by the COLINK function. For COBOL, the CALLed program must use the COBOL LINKAGE-SECTION to receive passed arguments.A maximum of eight parameters can be specified.

> **Note:** For PL/I, the declaration of entry point for the COLINK and COXCTL function should not be ASM, unless the target program is using Assembler linkage conventions [that is, Assembler, COBOL, or PL/I with PROC options (MAIN)]. If multiple target types are to be used, create a copy of the COLINK subroutine with a different name.

Return Codes

The COLINK function itself does not give a return code. Return is dependent upon the application program identified by the COLINK function.

Abends

An abnormal termination may occur during execution of the COLINK function. Possible causes include:

- The *name* argument is invalid;

- Not enough storage is available to load the program identified by the COLINK function;

- The COLINK program is not found either in storage or in the COMPLIB load library chain;

- A disk error occurred;

- A security violation occurred;

- An attempt was made to load a planned overlay program using COLINK.

COLOAD Function

The COLOAD function is used to load a program into the thread region area of an application program. The program to be loaded can reside:

- in the resident area of the operating system, or:

- in the resident area of Com-plete, or:

- in the Com-plete COMPLIB load library chain.

If a copy of the COLOAD program resides in the resident portion of the operating system or the resident portion of Com-plete, the program is not loaded into the application program thread region. In this situation, any COLINK or COXCTL functions for the designated program uses the resident copy of the program. For z/OS systems only, planned overlay programs cannot be specified in a COLOAD function.

The COLOAD function is normally used in conjunction with one or more subsequent COLINK functions. This technique avoids unnecessary loading of new copies of the same loaded program with each call to the COLINK function.

Each COLOAD function that successively loads a program into the application program thread region is associated with a use count by Com-plete. This use count is used to maintain the load status of the program. The use count is incremented by one for each successful COLOAD function and decremented by one for each successful CODEL function. When the use count becomes zero, the program is automatically deleted from the application program thread region, if it is not in use by an active COLINK or COXCTL function.

If more than 32,767 COLOAD functions are successfully executed with no intervening CODEL functions, the use count becomes negative. In this situation, the COLOAD program cannot be deleted, either automatically or with CODEL.

The program to be loaded can optionally be loaded using an SVC LOAD macro statement in an Assembler-written subroutine. If the program has previously been loaded into the application program thread region using either the COLOAD function or the SVC LOAD macro, the use count for the load is incremented by one. In addition, the loaded copy of the program is used to satisfy any additional load type request (that is, COLOAD, COLINK, COXCTL, CODEL, SVC LOAD, SVC LINK, SVC XCTL, SVC DELETE).

Format

The format for using the COLOAD function is:

```
COLOAD (retcode,name)
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
name	Required.
	An eight-byte alphanumeric field containing the name of the program to be loaded. The name must be left-justified and padded with blanks.

Return Codes

The following return codes are issued by the COLOAD function:

0	No errors.
4	The program to be loaded was not found in the COMPLIB load library chain or in memory.
8	An I/O error occurred while loading the program.
12	Not enough memory was available in the thread region to load the requested program.
16	A request was made to load a planned overlay type program. Planned overlay programs cannot be loaded.
20	A security violation occurred.

Abends

An abnormal termination may occur during execution of a COLOAD function. Possible causes include:

- The *name* argument is invalid;

- Sufficient storage is not available to build the control block required to successfully complete the load request.

COXCTL Function

The COXCTL function is used to pass control to a specified program. The program to receive control can reside in the resident area of the operating system, the resident area of Com-plete, or the application program thread region area. The program indicated by the COXCTL function logically replaces the calling program. The COXCTL function cannot be used to pass control to an z/OS planned overlay program.

If the program that issues the COXCTL function was given control by a COLINK function, the COXCTL program returns control to the program issuing the COLINK; otherwise, termination of the COXCTL program is absolute. Termination and/or transfer of control is generated in one of two ways:

- Execution of a STOP RUN statement (COBOL) or RETURN statement (PL/I);

- Execution of an EOJ function.

If the COXCTL program is not in memory and has not previously been loaded using a COLOAD or SVC LOAD, Com-plete loads the program into the application program thread region area. The program using the COXCTL function is deleted from the thread and is overlayed by the COXCTL program. Any arguments passed to the COXCTL program are destroyed by this overlay if they reside in the calling program.

Format

The format for using the COXCTL function is:

```
COXCTL (name[,arg1]...[,argn])
```

name	Required. An eight-byte alphanumeric field containing the name of the program to which control is to be given. The name must be left-justified and padded with blanks.
argn	Optional. Default: None.Any parameter(s) to be passed to the COXCTL program.For COBOL, the called program must use the COBOL LINKAGE-SECTION to receive passed arguments. For PL/I, standard linkage conventions are used to pass arguments.

> **Note:** For PL/I, the declarations of entry point for the COLINK and COXCTL functions should not be ASM, unless the target program is using Assembler linkage conventions [that is, Assembler, COBOL, or PL/I with PROC options (MAIN)]. If multiple target types are to be used, create a copy of the COXCTL subroutine with a different name.

Return Codes

There are no return codes associated with the COXCTL function. The COXCTL program does not return control to the calling program.

Abends

An abnormal termination may occur during execution of a COXCTL function. Possible causes include:

- The *name* argument is invalid;

- Sufficient storage is not available in the application program thread region to contain the COXCTL program;

- The COXCTL program was not found in the Com-plete program library;

- A disk error occurred;

- A security violation occurred;

- An attempt was made to COXCTL to a planned overlay program.

FETCH Function

The FETCH function is used to fetch a program into the application program thread, pass control to the fetched program, and (optionally) pass information to the fetched program. The program being fetched can reside:

- in the resident area of the operating system, or:

- in the resident area of Com-plete, or:

- in the Com-plete COMPLIB load library chain.

The FETCH function reinitializes the thread and establishes the program attributes of the program being fetched. Note that the thread lock number assigned to the program being fetched may be different from that of the calling program.

After execution of the FETCH function, the application program being fetched will receive control at its entry point. If data is being passed to the fetched program, the first Com-plete terminal I/O function executed by the fetched program must be a READ function to receive that data. If a terminal I/O function other than READ is issued, the passed data is destroyed.

The fetched program is executed exactly like an initially called program with the exception that the READ functions READS and READM cannot be used as the first READ function to be executed, since the data to be read is in translated format.

Format

The format for using the FETCH function is:

```
FETCH (name[,area,length])
```

name	Required. An eight-byte alphanumeric field containing the name of the program to be fetched. The name must be left-justified and padded with blanks.
area	Optional. Default: No data will be passed.A buffer area in the application program thread region of the calling program that contains information to be passed to the fetched program.
length	Optional. Must be specified if area is specified. Default: None.A binary halfword containing the length of the data identified by the area argument that is to be passed to the fetched program. The value specified by length cannot exceed 4094 bytes.

When the *area* and *length* arguments are used in the FETCH function, the amount of data specified is placed in a Com-plete terminal I/O buffer associated with the terminal in use. When the fetched program receives control, the information in this buffer can be obtained by issuing a terminal READ function; however, the terminal READ function must be the first terminal I/O function issued by the fetched program, or the contents of the buffer are lost.

Return Codes

There are no return codes associated with the FETCH function. Standard linkage conventions are followed by placing the address of the fetched module in register 15.

Abends

An abnormal termination may occur during execution of a FETCH function. Possible causes include:

- An invalid *length* argument was specified;

- The *name* argument specifies an item not found in the program library;

- An invalid *area* argument was specified;

- A protection exception has occurred;

- A disk I/O error occurred;

- An attempt was made to fetch to a planned overlay type program.

LOAD Function

The LOAD function is used to load a table or module into the application program area and (optionally) pass control to it. The table or module to be loaded must reside in the Com-plete COMPLIB load library chain.

The table or module to be loaded must be small enough to be loaded into the application program thread region of the calling program. The catalog attributes, with the exception of the region specification, must be the same.

After execution of the LOAD function, the application program can receive control at the instruction immediately following the LOAD function or, if loading a module, can optionally pass control to

the module. If the table or module to be loaded does not exist in the program library, the LOAD function terminates abnormally.

The LOAD function is used when loading a table or module into the application program thread region and passing control to the instruction immediately following the LOAD function.

The LOADT function is used when loading a module into the application program thread region and passing control to that module.

Format

The format for using the LOAD function is:

```
LOAD[T] ([epret],name[,area][,length])
```

epret	Required for LOAD.A fullword where Com-plete returns the entry point address of the module loaded.
name	Required. An eight-byte alphanumeric field containing the name of the table or module to be loaded. The name must be left-justified and padded with blanks.
area	Optional. Default: The load point of the calling program.A double word-aligned buffer area in the application program thread region where the table or module identified by the name argument is to be loaded.
length	Optional. Default: The physical size of the table or module being loaded.A binary halfword containing the length of the table or module to be loaded. The storage area defined by the area and length parameters or their defaults must fully reside inside an area of storage available to the calling program.

Return Codes

There are no return codes associated with the LOAD(T) function. Upon return from the LOAD function, register 15 contains the entry point address of the module loaded.

Abends

An abnormal termination may occur during execution of a LOAD function. Possible causes include:

- An invalid *length* argument was specified;

- The *name* argument specifies an item not found in the program library;

- An invalid *area* argument was specified;

- A protection exception occurred;

- A disk I/O error occurred;

- An attempt was made to load a planned overlay type program;

- The item being loaded does not fit into the area specified or defaulted.

SCHED Function

The schedule (SCHED) function allows an application program to cause a conversational program to be started at another terminal, as if there had been input from that terminal. This function is used by Com-plete graphic support to schedule printing at graphics printers.

Format

The format of the SCHED function is:

```
SCHED (retcode,name,area,length,destlist,listlen,flag)
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
name	Required. Label of an area containing the name of the application program to be started. Must be 8 bytes and padded with blanks.
area	Required. The label of a data area containing the input data to be presented to the SCHEDed program when it issues a terminal read.
len	Required. The label of a halfword data area containing the length of the input data.
destlist	Required. The label of a data area containing the names or numbers of the terminal(s) to which the SCHED function is directed.Note that each entry in destlist must be eight characters long, left-justified, and padded to the right with blanks.
listlen	Required. The label of a halfword data area containing the number of terminals in the destination list.
flag	Required. The label of a one-byte data area containing a flag to control processing. The userid of the application issuing the SCHED function is propagated to the SCHEDed task. If the flag byte is x`80', the terminal to which the SCHED function is directed will remain logged on to that userid after termination of the SCHEDed program. If the flag is not x`80', the terminal will be logged off after completion of the SCHEDed program.

Return Codes

The following return codes are issued by the SCHED function:

0	Normal return.
4	Program not found.
8	A security violation has occurred. This can occur if the invoker of the SCHED function does not have access to the requested program, or if the terminal on which the program should be run does not have the appropriate receive class codes.
12	An unrecoverable I/O error has occurred.
16	Too many receiving terminals were specified.
20	An invalid destination code was specified.
24	A negative segment length was specified.
28	The message text was too long. This return code is provided for Class 16 messages only.
32	Not enough storage for request.

Abends

Abnormal termination may occur during the execution of a SCHED request. Possible causes include:

- An incorrect number of parameters was specified;

- An invalid *area* or *len* argument was specified;

- An invalid *destlist* or *listlen* argument was specified;

- No eligible thread exists in which to run the scheduled program.

IV Message Switching and Printout Spooling

This part of the Application Programming documentation covers functions relating to message switching and printout spooling functions. This includes NSPOOL spooling with Natural Front-End which runs Natural under Com-plete

This information is organized under the following headings:

Message Switching/Printout Spooling

NSPOOL - Printout Spooling With Natural Front-End

9   Message Switching/Printout Spooling

Overview

The Com-plete message switching and printout spooling functions enable online and batch application programs to send messages or printout data sets to one or more terminals. The receiving terminal(s) can be either a hard copy terminal device or a soft copy (display) terminal device.

Messages and printout data sets are both processed by the Com-plete message switching task, although separate functions must be specified for each type of request.

The distinction between a message and a printout is normally the number of output lines and/or the type of terminal device(s) to receive the output.

- *Messages*
  Messages are considered to be text that is not column- or line-oriented and that is comprised of words, sentences, and paragraphs. Messages are also assumed to contain a relatively small amount of data. Because of this, Com-plete will sometimes write a message to a terminal in a slightly different format than it was sent. For instance, Com-plete attempts to start a line between words so that words are not broken. 3270-specific WCCs and control attributes are not valid as data.

- *Printouts*
  Printouts are assumed to contain formatting and spacing that cannot be altered; they are also assumed to contain a large amount of data. Because of this, lines are always written as is. If the receiving terminal's line length is too short, the output line will be truncated.

When a message switching or printout spool function is initially invoked, Com-plete creates a buffer core queue in order to maintain tracking control over the message or printout. When the message or printout is successfully received, the core queue is freed, and the space occupied becomes available for use. Although core queues are relatively small in size, a large number of messages or printouts requires a large number of core queue elements.

The core queue elements are used by Com-plete to enable message or printout restart/recover in case of an abnormal system failure.

The message switching and printout spooling functions of Com-plete are available to both batch and online programs. The message or printout awaiting receipt may be monitored by use of the Com-plete online utility program USPOOL, which is described in the Com-plete Utilities documentation.

Message Switching

The message switching facilities of Com-plete enable an application program to send messages to one or more terminals in the Com-plete network. The application program can be an online or batch program.

Messages can also be sent from a given terminal to one or more terminals without using an application program. This facility is provided by the Com-plete online utility program UM, which is fully documented in the Com-plete Utilities documentation. Message switching performed by an application program is accomplished by use of the MESGSW function.

When a message is sent to a terminal, it is by default copied onto a disk file containing queues of messages for each terminal. Messages can be recalled from this file by use of the UM utility program at any time, as long as they have not been deleted.

Message Segmentation

Messages are formed in pieces called segments. Each segment cannot exceed 32,767 bytes in length. There is no limit to the number of message segments that comprise a given message, although the size of the message queue file itself is a limiting factor. Before a segmented message is written to the receiving terminal device, the segments are linked together by Com-plete, and the message is sent as a single unit.

All messages, regardless of content, are treated as text when they are displayed at the receiving terminal. They are printed or displayed using the maximum line length of the receiving device. A word that will not fit at the end of a line is moved to the beginning of the next line. All blanks at the end of a message are ignored.

Limited message output formatting can be accomplished by embedding the Com-plete terminal new line symbol in the message text. The new line symbol is X`15'. This symbol is device-independent and produces the same results regardless of the device type to which the message is sent.

Destination Codes

When a message is sent, information stating where the message is to be received must be included. The terminal(s) to receive the message can be specified by referencing one or more destination codes. Each destination code is a TID number, a destination code that represents one or more TIDs, or a user ID. Destination codes are defined by the installation to contain convenient groupings of TIDs. Procedures for changing and adding destination codes are also defined by the installation. Destination codes used in a message switching request are converted into TID numbers, and the message(s) is sent to the appropriate terminals. If the destination code is a user ID, the notation "U=user-id" can be used to distinguish from a group name.

The maximum number of terminals that can receive a message via the MESGSW function is 100, even if a destination code is used. This maximum number restriction can be circumvented by sending multiple copies of the same message, specifying 100 or fewer receiving terminals each time.

Class Codes

Security restrictions and other information about a message are designated by class codes. There are two categories of class codes: security class codes (those numbered 1 through 4), and other class codes (those numbered 8 through 16). Each message must have at least one security class code assigned, or the message request will be unsuccessful.

Each terminal in the Com-plete network has two sets of security class codes assigned to it: one that defines which message classes it can send, and one that defines which message classes it can receive. If the accounting option is used, class codes are assigned through your User ID. If the accounting option is not used or if you have not logged on to Com-plete, class codes are assigned through the terminal definition table.

Messages must have security class codes assigned. Com-plete checks to be sure that the sending terminal is authorized to send a specific message, since the class code(s) assigned to a message must be among the sending class codes assigned to the sending terminal. Com-plete also determines that the assigned class code(s) is(are) included in the receiving class codes of the terminal(s) to which the message is sent. If the class codes are not compatible, the message is not sent, and in some cases, a security violation is logged to the Com-plete logging device.

Several class codes (those numbered 8 through 16) cause certain special operations to be performed as a message is sent. These operations and all the class codes are explained in the following figure.

Class codes 1 through 4 are used for security checking. Note that every message or printout must have at least one of these classes.

Class codes 5 through 8 are not currently used and are ignored when specified.

Class 1	Standard message class. Messages with this class assigned do not interrupt a terminal while it is in conversation with a program.
Class 2	Urgent message class. Overrides the MESSAGE DISABLED status of a terminal and causes the message to be displayed immediately at the receiving terminal. The receiving terminal is interrupted if it is in conversation with a program. If the receiving terminal has the audible alarm feature, the audible alarm will sound.
Class 3	Special purpose class code. The message interrupts a terminal in conversation. These messages are always restarted from the beginning if Com-plete is reinitialized before they are sent successfully.

Class 4	Reserved for Com-plete logged messages. No application program or terminal can initiate a Class 4 message. These messages are exempt from restart in the event that Com-plete is reinitialized before they have been received successfully.

The following class codes are referred to as other class codes. They cause special services to be performed as a message is sent.

Class 8	Reserved for future usage.
Class 9	Reserved for future usage.
Class 10	Reserved for future usage.
Class 11	Reserved for future usage.
Class 12	Causes a message to be deleted from the message queue file after 30 minutes, if the message cannot be sent to the receiving terminal. These messages are exempt from restart in the event that Com-plete is reinitialized before they have been received successfully.
Class 13	Causes a message to be written without the standard message header that normally accompanies all messages.
Class 14	Standard message class; no special action taken. These messages are exempt from restart in the event that Com-plete is reinitialized before they have been received successfully.
Class 15	Causes the audible alarm to be sounded when the message is written to the terminal, if the receiving terminal has the audible alarm feature.
Class 16	Prevents a message from being queued to the message queue file on disk. Instead, the message remains in main storage until it is successfully sent. The length of a message using this class code is limited to the amount of text that can be contained in one message buffer. This amount can be calculated by doubling the number of terminals to receive the specified message and subtracting this amount from 240.For example, if there were three receiving terminals, a Class 16 message could be a maximum of 234 characters long; however, the message would not be requeued upon restart.

Message Routing

With the exception of Class 13 messages, each message received has a standard header containing the message identification number, date, time the message was written, and TID number of the terminal from which the message was sent. The following display illustrates a typical header message.

```
MSG ID:         133, SENT 10/17/86 AT 1106, FROM TID 17
SAMPLE MESSAGE HEADER DISPLAY
```

If this header is not desired or if the five new line symbols appended to a message sent to a hard copy device are not wanted, the message to be sent must be assigned Class code 13.

Messages sent to hard copy devices are written continuously. Messages sent to CRT devices require acknowledgement after being displayed. Acknowledgement is accomplished by pressing **ENTER**. If the message is not acknowledged immediately, the next null input entry will be considered an

acknowledgment. In addition, if an application program is in use at the receiving terminal, any screen formatting in use may be destroyed. The specific procedures for recovering a screen format are application-dependent, and the specific application in use at the time of the interrupt must be referenced for recovery procedures. Note that, regardless of the acknowledgment procedures, the application in use is not terminated, only interrupted.

If a message sent is longer than the buffer size for the receiving terminal and the display is not fully displayed, an asterisk (*) is the last character displayed. The remaining portion of the message can be displayed by pressing **ENTER**.

Messages sent to terminals cannot be received immediately for the reasons indicated below. When this occurs, all messages not received are queued to the destination terminals. Since Com-plete uses buffers to process terminal communication messages, a potential system problem exists. A large queue of messages that cannot be sent could potentially saturate the buffer pool, disabling all terminal communications and causing a system lockout condition. This situation can be avoided if application programs that send messages use the GETCHR function of Com-plete prior to using the MESGSW function to interrogate the message receipt status of the destination terminal(s). If the number of messages or printouts queued to a destination terminal exceeds a specified value, the application program may issue a warning message to the appropriate user, refuse to send additional messages until the number of queued messages is acceptable, or take some other corrective action.

Three common situations in which messages sent to a terminal are *NOT* automatically displayed are:

- The terminal is MESSAGE DISABLED. Occasionally when messages are sent to your terminal, they may interrupt and destroy whatever data is being entered at the time. To prevent this, the you can set the terminal to disabled status for receipt of messages; however, disabled status does not prevent Class 2 messages from being received.

- A conversational program is executing at the terminal. If an active program is currently in use at the receiving terminal, messages are placed in a special message queue for that terminal. The messages are received when the application program terminates. This feature does not apply to Class 2 or 3 messages.

- The previous message was not acknowledged. If a previously sent message has not been acknowledged, additional messages are queued. This condition can occur only with CRT devices.

When a terminal is enabled for receipt of messages (for example, ending a conversational program, removing the MESSAGE DISABLED status), any messages that have been placed in the message queue are available for display. For CRT terminals, acknowledgment of each message queued to the terminal causes the next message to be displayed until all have been displayed and acknowledged. Messages queued to a hard copy terminal are printed automatically when the terminal is made available to receive them.

When a terminal is MESSAGE DISABLED or in conversation, all UM utility commands can be used to display specific messages or message information without altering the disabled or conversational status on termination of any active program.

Alternate Terminals

Alternate terminals are terminals designated to receive message output for terminals that are inoperative, currently in conversation with an online program, or disabled for message receipt. Alternate terminals are assigned either through the terminal definition table (TIBTAB) or via the ALTERNATE command function of UM.

If a terminal has an alternate terminal assigned and cannot immediately receive a message, any message sent to it will automatically be displayed or written to the alternate terminal. Messages rerouted to an alternate terminal are *not* queued to the original receiving terminal.

Disabled Terminals

Disabled terminals are terminals that have been disabled for receipt of messages by use of the DISABLE command function of UM.

When messages are sent to a terminal, they may interrupt and destroy whatever data may have been entered at the time the message was sent. To prevent this, the DISABLE command function of UM is used. When messages (except those sent with Class 2) are sent to a disabled terminal, they are routed to an alternate terminal, if one has been assigned. If an alternate terminal has not been assigned, the messages are placed in the message queue for the disabled terminal. Class 2 messages override a DISABLED status and are displayed immediately.

Messages sent to a disabled terminal, if not rerouted to an alternate terminal, are automatically displayed when the terminal is once again enabled to receive messages.

Inoperative Terminals

Inoperative terminals are terminals that are either turned off, malfunctioning, or write-inhibited via a switch.

Com-plete handles inoperative terminals in a manner similar to the way it handles DISABLED terminals: messages sent to these terminals are rerouted to a designated alternate terminal, if one has been assigned. If an alternate terminal has not been assigned, the messages are placed in the message queue for the inoperative terminal.

Messages sent to an inoperative terminal, if not rerouted to an alternate terminal, are automatically displayed when the terminal is once again able to receive messages.

Message Recovery

Messages sent to hard copy terminals are immediately written, if the terminal is in ready status and no other messages are queued to the terminal.

The Com-plete message switching facility provides automatic message restart in the event that terminal output is interrupted. Message restart is performed on a checkpoint basis. Each message sent to a terminal is initially queued to the message switching file queue that resides on disk. Messages residing on the message file queue are sent, or written, to the destination terminal on an availability basis. As messages are written, a core queue checkpoint is taken to indicate the status of the output being written. For large messages, this checkpoint is taken at the completion of each full page of output. If output is interrupted (for example, by pressing **STOP**), output resumes at the last checkpoint when the terminal is again made ready (for example, by pressing **START**).

If a system failure occurs, all message core queue checkpoint records are destroyed. When Com-plete is again initialized, various message restart options are available, the default being recovery of message output from the beginning of the output message. The system programmer responsible for Com-plete maintenance should be consulted for more details on the message recovery options available after a system failure.

Message Switching Control Block (MESGCB)

Before an application program can send a message, it must create and initialize a Message Switching Control Block (MESGCB). The MESGCB is a working storage area in the program containing information needed by Com-plete to control processing of the message.

Three types of information exist in the MESGCB:

- Class codes;
- Segment flags;
- Message identification numbers.

You can set class codes before sending the message.

The segment flag is also set by the application program before sending the message. If the message is to be treated as an entire message, the segment flag must be initialized to an L. If the message is to be treated as a segment of the entire message, the segment flag must be initialized to a space (X`40'). For segmented messages, the segment flag must be initialized to an L when the last segment has been created and is ready to send.

The message identification number must always be initialized to spaces. When the message is sent, Com-plete uses this field and initializes it with the message number assigned to the message. If the message is being sent in segments, the message number is set by Com-plete in this field after using the MESGSW function to send the first segment.

The format and content of the MESGCB are illustrated in *Message Switching Control Block (MESGCB)*.

A DSECT of the MESGCB can be created using

```
name MCALL MESGCB
```

MESGSW Function

The MESGSW function is used by the application program to send a message or a message segment. The message switching control block must have been defined and initialized prior to execution of the MESGSW function.

Format

The format for using the MESGSW function is:

```
MESGSW (retcode,mesgcb,area,length[,destination][,number])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
mesgcb	Required. The message switching control block to be used when sending this message. This control block must have been previously defined and initialized in the application program.
area	Required. A buffer area in the application program containing the message or message segment to be sent.
length	Required. A binary halfword containing the length of the message or message segment to be sent.
destination	Optional. (Required for the first segment of a message.)Default: None. This keyword is ignored for message segments that are not the first segment. A table defined in the working storage area of the application program. This table contains destination codes and/or TIDs eligible for receipt of messages. Each code in the table must be eight bytes long, left-justified, alphanumeric in format, and padded to the right with blanks. In some cases, you may need to use 10-bytes long table entries. For example, to send to 7 or 8-byte user IDs using the U=userid notation. Add 32768 to the actual number of destinations (see the "number" parameter for details) to advise Message Switching to interpret the destination table as having 10-byte entries. This is equivalent to setting the high-order bit of the number halfword to one in Assembler language programs.
number	Optional. (Required for the first segment of a message.) Default: None.A binary halfword containing the number of destination codes (defined in the previous table) that are to be used.

Return Codes

The following return codes are issued by the MESGSW function:

0	Normal return.
4	A message segment was requested. The segment requested was not the first segment, and the first segment cannot be found. Probable MESGCB control block error.
8	A security violation has occurred, caused by invalid class codes.
12	An unrecoverable I/O error has occurred.
16	Too many receiving terminals were specified.
20	An invalid destination code was specified.
24	A negative segment length was specified.
28	The message text was too long. This return code is provided for Class 16 messages only.

Abends

An abnormal termination may occur during execution of the MESGSW function. Possible causes include:

- An invalid control block was specified;

- An invalid *area* or *length* argument was specified;

- An invalid *number* argument was specified;

- A Class code of 4 was specified. Application programs cannot use Class code 4.

Printout Spooling

The Com-plete printout spooling functions enable an online or batch application program to generate print output destined to be printed on a hard copy terminal. More than one program can generate printouts destined for the same device, and individual printout data sets are spooled separately. The outputs are not intermixed.

Print output is created by the application program one statement, or line, at a time; however, the print output itself occupies a logical printout spool data set and is not available for printing on a terminal(s) until the data set has been queued to the terminal(s). The functions available for printout spooling requests enable the application program to allocate the printout spool data set, write the print output statements to the data set, and queue the data set for scheduled printing. The printout spool functions available for use by an application program are summarized in the following table.

Function	Description
PSOPEN	Allocate a printout spool data set.
PSPUT	Write statements to the printout spool data set.
PSCLOS	Queue the printout spool data to the destination terminal(s).

In addition to using the Com-plete printout spool functions to generate print output, the application program must also define a Printout Spool Control Block (PSCB) in the working storage area of the application program.

The information in the PSCB is used to identify the printout and to control the output form and the logical output driver.

Specifying a logical output driver enables you to modify the listing during output. This facility can be used to create extra output lines (for example, header, trailer) for each printout. In the event of changes, you need only to change the logic of the logical output driver instead of changing all their application programs. See the supplied modules LDRVSAMP and IPDSDRV in the Com-plete source for examples.

Since the printout spool facility of Com-plete also utilizes the message switching facility of Com-plete, printout data sets must be assigned class codes in order to determine authorization for both the receipt of printout spool data sets and the ability to generate printout spool data sets. The PSCB is referenced by each of the printout spool functions and is used when defining the required authorization class codes.

The terminal(s) destined to receive a given printout spool data set is also defined in the working storage section of the application program in a table of destination codes or TIDs. Since print output is spooled on a printout spool data set basis, this table is referenced only by the PSOPEN function when defining the printout spool data set.

Destination Codes

The terminals to which spooled printouts are destined are specified using TIDs defined in a table or by referencing one or more destination codes also defined in a table. Both TIDs and destination codes can be used in the same table. Each entry is defined as an alphanumeric entry, left justified, and padded with blanks.

The destination ID "SYSOUT" is a special-purpose destination used to route a printout to the operating system spool. The default output class in the operating system is "A". Note that the output class can be modified by specifying SYSOUT=x as destination, where *x* represents the output class.

Each destination code represents one or more TIDs and is defined by the installation for convenient groupings of terminals. Procedures for changing and adding destination codes are also set by the installation.

Destination codes defined in an application program are converted to TIDs, and spooled printout data sets are sent to the appropriate terminals. Multiple copies of a printout spool data set can be

sent by specifying the same destination entry more than once; however, the maximum number of receiving terminals per printout spool function cannot exceed 100 terminals.

Class Codes

Security restrictions for printout spool data sets are designated by the message switching class codes. Each terminal in the Com-plete network has two sets of security class codes assigned to it: one defines which class codes are valid for sending, and one defines which class codes are valid for receipt. These class code assignments are made on a terminal basis and, in the case of conversational type terminals, can be overridden by the user ID of the terminal user after a *ULOG ON request is issued.

A printout spool data set must have a class code associated with it. Com-plete checks to be sure that the terminal from which the application program is called is authorized to send the printout spool data set. The class code(s) assigned to the printout spool data set must be among the sending class codes of the terminal in use.

Com-plete also determines that the class code(s) assigned to the printout spool data set is included in the list of receiving class codes of the terminal(s) to which the printout data set is sent. If the class codes are not compatible, the printout data set is not sent.

The class codes available for use with the printout spool facility of Com-plete are Class codes 1 through 7. These class codes are fully described in the previous section *Message Switching*. Class codes 8 through 16 are reserved exclusively for the message switching facility of Com-plete and should not be used with printout spooling requests.

Disabled, Inoperative, and Alternate Terminals

A terminal destined to receive a printout spool data set can be disabled, inoperative, or have an alternate assigned. The definition of these conditions and the resulting routing conditions that occur are fully described in the previous section *Message Routing*.

Printout Spool Control Block (PSCB)

The Printout Spool Control Block (PSCB) is defined in the working storage section of the application program. One PSCB must be defined for each printout spool data set to be used by the application program.

The PSCB for a given printout spool data set is referred to by each printout spool function requested for the data set and used to contain information needed by Com-plete to control the routing of the printout data set. The following six types of information are contained in the PSCB:

- Listname;
  The listname must be established by the application program before use of the PSOPEN function.
- Form name.

- Logical output driver

- Class codes;
  The class codes settings are established by the application program before use of the PSOPEN function.

- Logical statement length;
  The logical statement length identifies the maximum line length in a print output. You can set lower values in the PSPUT parameter list. Note that the maximum logical statement length must be between 8 and 256 bytes.

- Printout identification numbers;
  The printout identification number is a sequential number assigned by Com-plete to the printout spool data set. Note that this number must not be modified by the application program or an abnormal termination will occur in the application program.

The format of the PSCB is illustrated in **Printout Spool Control Block (PSCB)**.

A DSECT of the PSCB can be created using

```
name MCALL PSCB,LRECL=nn
```

where nn is the length of the records to be written to the spoolfile.

PSOPEN Function

The PSOPEN function provides the information needed by Com-plete to create a printout spool data set. The application program must initialize a PSCB for each printout spool data set for which the PSOPEN function will be used.

Format

The format for using the PSOPEN function is:

```
PSOPEN (retcode,pscb,codes,numcodes)
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
pscb	Required. Specifies the printout spool control block defined for the printout spool data set. This control block must have been previously initialized by the application program.
codes	Required. Each entry in the table must be an eight-byte alphanumeric field, left-justified, padded to the right with blanks.Specifies a table of destination codes and/or TIDs to be used to identify which terminal(s) are to receive the printout spool data set when a PSCLOS function is issued.

numcodes	Optional.
	Default: 1Specifies a binary halfword containing the number of entries to be used from the codes table.

The maximum number of receiving terminals that can be specified by the PSOPEN function is 100. If more than 100 terminals are to receive the printout spool data set, then the PSOPEN function must be reused after the PSCLOS function is used. Alternatively, more than one PSOPEN function can be used simultaneously using an additional PSCB for additional PSOPEN functions.

Return Codes

The following return codes are issued by the PSOPEN function:

0	Normal completion.
4	A PSCB control block error has occurred.
8	A security violation has occurred. The terminal in use is not authorized to send the printout spool data set being requested for creation.
12	The class codes specified were invalid.
16	Too many receiving terminals were specified.
20	One or more destination codes were invalid.
24	An invalid record length was specified.

Abends

An abnormal termination may occur during execution of the PSOPEN function. Possible causes include:

■ An invalid PSCB control block was specified;

■ An invalid number of destinations was specified.

PSPUT Function

The PSPUT function is used to output, or write, a record from a working storage buffer area in the application program to the printout spool data set named by the PSCB argument. Note that the record is spooled to the data set. The record is not immediately sent to the destination terminal(s).

Format

The format for using the PSPUT function is:

```
PSPUT (retcode,pscb,area[,length])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
pscb	Required. Specifies the printout spool control block assigned to the printout spool data set. This control block must have been previously defined by the application program and must have been opened with the PSOPEN function.
area	Specifies a buffer area in the working storage area of the application program containing the record to be written to the data set. The first character of this field must be one of four carriage control characters:

	blank	Advance one line before printing.
	0	Advance two lines before printing.
	-	Advance three lines before printing.
	1	Advance to a new page before printing.

length	Optional. Specifies a two-byte unsigned binary field containing the length of the text in the area. If this length is smaller than the logical line length in the PSCB, the line is padded with blanks. If it is greater, it is truncated. 0 is not a valid value.

Return Codes

The following return codes are issued by the PSPUT function:

0	Normal return.
4	A PSCB control block error occurred.
12	An unrecoverable I/O error occurred.
24	An invalid logical record length was specified.

Abends

An abnormal termination may occur during execution of the PSPUT function. Possible causes include:

- An invalid PSCB control block was specified;
- The PSCB control block was not opened with the PSOPEN function;
- An invalid *area* argument was specified.

PSCLOS Function

The PSCLOS function is used to logically close the printout spool data set identified by the PSCB argument. Use of the PSCLOS function causes the identified printout spool data set to be queued for printing to the destination terminals.

Format

The format for using the PSCLOS function is:

```
PSCLOS (retcode,pscb)
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
pscb	Required. Specifies the printout spool control block that identifies the printout spool data set to be queued for output. This control block must have been previously defined and opened by the application program.

Return Codes

The following return codes are issued by the PSCLOS function:

0	Normal completion.
4	A PSCB control block error occurred.
12	An unrecoverable I/O error occurred.

Abends

An abnormal termination may occur during execution of the PSCLOS function. A possible cause is that the PSCB was not opened.

10 NSPOOL - Printout Spooling With Natural Front-End

This section describes the NSPOOL utility, Com-plete's printout spooling facility for sites that run Natural under Com-plete.

Overview

CSPOOL is the Application Programming Interface (API) to Com-plete's spooling utility and provides NSPOOL functionality. The definition of the API, as well as customization information is given in section *Customization*.

NSPOOL is an example Natural application distributed on the Com-plete installation medium and can be modified to provide a site-specific printout spooling facility. NSPOOL permits the flexible management and distribution of output to any online printer in the TP network.

Many applications require output to be printed on special forms. A problem often arises when different applications require different forms to be mounted on the same printer at the same time. With NSPOOL, you can create printouts on virtual printers that are not currently active in the system and subsequently route the printouts through the TP network to the physical printer where the appropriate forms have been mounted.

In addition, NSPOOL can be used to provide relevant information pertaining to each printout in the system, thus providing a comprehensive overview of all queues for all printers. You can also display the contents of a printout before requesting a print operation.

NSPOOL also supports the operation of online printers using commands such as DISPLAY STATUS, HALT, and RESET. In addition, you can route any output from the online queues to the system spool by using the special destination SYSOUT.

NSPOOL is completely menu-driven and provides the capability of full screen data entry. An online HELP facility is also available to assist you.

Natural Front-end

NSPOOL provides a front-end written in Natural. This has implications both in the areas of security and customization.

- With Natural Security, user access to printers, printer groups and functions on printouts can be controlled and maintained by the system administrator.

- The Natural-based user interface allows site-specific adaptation to corporate standards, as well as the integration of site-specific functions and features.

> **Note:** The system programmer for your Com-plete installation also has the option of restricting access to any particular NSPOOL/USPOOL function via the UUSPL0 exit. Note that the exit takes priority over any definitions made in Natural Security.

Natural Security Definitions

1. Under Natural, you must define NSPOOL as Library to Natural Security using the Library Maintenance facility. Specify the following user exit in the library definition:

```
USEREXIT: SPSE01-N
```

2. Additionally, enter SYSSEC as Steplib using the Additional Options (see the section *Additonal Options* in *Library Maintenance* in the *Natural Security* documentation). Note that you can only enter SYSSEC on entry level 1-8 (do not use entry level 9).

3. Using the User Maintenance facility in Natural Security, you must link each user to the Library NSPOOL. The link must be of type SL (special link), and you must specify user exit SPSE01-N in the special link.

NSPOOL Definitions and Authorizations

The NSPOOL definitions immediately follow the Natural Security definitions. At the application level, printers can be organized into groups that can be referred to in user authorizations. User authorizations are made at the user level.

Printer Groups

Once NSPOOL has been defined as library (NAT31 and above), the next screen allows you to define up to five printer groups for the application NSPOOL , each containing up to five printers. This table can be modified subsequently using the Modify Application/Library function of Natural Security:

```
-----------------------PRINTOUT-SPOOLING-SECURITY-----------------------
                            APPLICATION DATA

   Application: NSPOOL                    Prt-Group: ___   Printer: _____
                                                                    _____
                                                                    _____
                                                                    _____
                                                                    _____


   Prt-Group: ___   Printer: _____     Prt-Group: ___   Printer: _____
                             _____                                _____
                             _____                                _____
                             _____                                _____
                             _____                                _____


   Prt-Group: ___   Printer: _____     Prt-Group: ___   Printer: _____
                             _____                                _____
                             _____                                _____
                             _____                                _____
                             _____                                _____

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
     HELP  ---   END   ---   UPD   ---   ---   ---   ---   ----  ----  ----
```

This table serves to make user authorization of printers more flexible: authorizing the user for one or more groups allows him/her to access the printers specified in the group(s).

Meaning of the input fields:

Prt-Group	Group identifier	
Printer	Printer name, or selection criteria to specify several printers:	
	*	specifies all printers
	prefix*	Specifies all printers whose names start with the prefix.

Press **PF5** to save the printer group definitions.
Having reached the "Application Data" screen via the special link, you can branch immediately to the user authorization for NSPOOL with **PF10** or **PF11** (the special link must already exist between user and NSPOOL).

User Authorization

Once the special link between the user and NSPOOL has been defined, the next two screens allow you to authorize the user to perform functions on specific printers in the printer overview and on specific printouts in the printout queue.

These definitions can be modified subsequently by redisplaying the special link definitions for the user and pressing **ENTER** to display the first Printout Spooling Security screen:

```
   ------------------------PRINTOUT-SPOOLING-SECURITY------------------------


   Application: NSPOOL    User: MBE      linked as: ADMINISTRATOR


   disallowed = D
   allowed    = A        Printernames      Functions
   -------------         ------------      ---------
   A                     *_____          *__  ___  ___  ___  ___

   _                     _____          ___  ___  ___  ___  ___
   _                     _____          ___  ___  ___  ___  ___
   _                     _____          ___  ___  ___  ___  ___

 Functions: *   - all following Functions
            PO* - Printer Overview  (PO, POS, PON)
            PP* - position Printout (PPT, PPB, PPA, PPR)  OPH - halt  Printer
            OP* - operate  Printer  (OPH, OPR, OPS, OPM)  OPS - start Printer
            OPM - mount    Form of Printer                OPR - reset Printer

 Note: Cancel current Printout (OPC) and flush all Queue-Entries (OPF) are
       controlled via the Function 'purge Queue-Entries (MQP)'.

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
     HELP  ---   END   ---   UPD   ---   ---   MORE  APPL  ----  ----  ----
```

The top of this screen shows the application name (NSPOOL), the user ID for which the definition is valid, as well as the user type.

The input fields are explained in the table on the next page:

Disallowed/ Allowed	Specifies whether the user is or is not allowed to perform the	
	function(s) specified in the Functions field for the specified Printer name. Possible options:	
	A:	user is authorized.
	D:	function is disallowed for the user.
Printername	Name of the printer or printer group for which the authorization is valid. You can either specify a printer name, a group name, or selection criteria as follows:	
	*:	authorization is valid for all printers.

	prefix*:	authorization is valid for all groups or printers whose names start with this prefix.NSPOOL first checks the printer group names for a match of selection criteria and if no match is found, printer names are searched.
Functions		Functions for which authorization is granted or denied. Available functions are listed in the lower half of the screen. The first two characters of the functions identify the function type (e.g., PP=Position printer), the third character identifies the function as described in the section Printer Overview below). You can specify any function value listed (*, PO* PP*, OP*, OPM, OPH, OPS, OPR, note that wildcard and prefix selection is possible). Note that any function value also includes the PO function. Note also that the values in brackets in the list of functions are valid only as a reference if your site wishes to program its own frontend.

Press **PF5** to save the printer overview function authorizations.

With **PF8** you continue to the List Queue function authorization table.

With **PF9** you can branch immediately to the printer group definition screen described above.

The List Queue function authorization table is illustrated on the next page:

The List Queue function authorization table:

```
   ------------------------PRINTOUT-SPOOLING-SECURITY------------------------


   Application: NSPOOL


   disallowed = D
   allowed    = A    Originator    Listnames    Functions
   --------------    ----------    ---------    ----------------------------
   A                 *_____      *_____     *__  ___  ___  ___  ___  ___
   _                 _____      _____     ___  ___  ___  ___  ___  ___
   _                 _____      _____     ___  ___  ___  ___  ___  ___
   _                 _____      _____     ___  ___  ___  ___  ___  ___


   Functions:   *   - all following Functions
               LQ* - list   Queue-Entries     (LQ, LQS, LQN)
               DP* - display current Printout (DP, DPN)
               MQ* - modify Queue-Entries     (MQU, MQC, MQM, MQP)
               MQU - update Queue-Entry
               MQC - copy   Queue-Entry
               MQM - move   Queue-Entry
               MQP - purge  Queue-Entry

   Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
        HELP  ---   END   ---   UPD   ---   UP    ---   APPL  ----  ----  ----
```

The input fields are explained in the following table:

Field	Meaning
Disallowed/Allowed	Specifies whether the user is or is not allowed to perform the
	function(s) specified in the Functions field for the specified Originator and Listname. Possible options: A: -- user is authorized. D: -- function is disallowed for the user.
Originator	User ID of the printout owner. You can either specify a user ID or selection criteria as follows:
	<table><tr><td>*:</td><td>authorization is valid for printouts belonging to any user ID.</td></tr><tr><td>prefix*:</td><td>authorization is valid for printouts belonging to any user whose ID starts with this prefix.</td></tr></table>
Listname	Name of the printout for which the authorization is valid. You can either specify a list or printout name or selection criteria as follows:
	<table><tr><td>*:</td><td>authorization is valid for all printouts according to the Originator specification.</td></tr><tr><td>prefix*:</td><td>authorization is valid for all printouts whose names start with this prefix, according to the Originator specification.</td></tr></table>
Functions	Functions for which authorization is granted or denied. Possible function values are listed in the lower half of the screen. The first two characters of the functions identify the function type (e.g., MQ=Modify Queue entries), the third character identifies the function as described in the section List Queue below). You can specify any value listed on the left of the function list (wildcard or prefix selection is possible). Note that any function also includes the LQ function. Note that the values in brackets on the right are valid only as a reference if your site wishes to program its own front-end.

Press **PF5** to save the printer overview function authorizations.

With **PF7** you return to the Printer Overview function authorization table.

With **PF9** you can branch immediately to the printer group definition screen described above.

Input Interdependencies

Note that if you fill one input field of an authorization table, you must also specify values in the other fields on the same line.

If you enter an authorization in the Printer Overview authorization table, you must also specify authorization parameters in the same line of the List Queue authorization table, otherwise default values take effect (see below). This means that if you enter an A in the authorization field, you must at least fill the function field of the Printer Overview table with PO* and the function field of the List Queue table with LQ*.

Conversely, if you delete an entry from the Printer Overview table, you must also delete the entry from the same line in the List Queue table.

Default Authorization

NSPOOL can be used with or without Natural Security.

If NSPOOL runs under Natural Security, the user's logon user ID is searched in Natural Security. If the user ID is not defined, the request for NSPOOL is rejected.

If the user ID is defined to Natural Security, authorization is assigned as follows:

If the user ID has a Special Link to the application, but no security entry is made in the link, the user is assigned a default authorization. The default authorization consists of the following functions:

- List all printers;
- List all printouts belonging to the user ID;
- Display any printout belonging to the user ID.

NSPOOL User Functions

To reach the NSPOOL Main Menu, enter the following command:

```
*NSPOOL
```

The following screen appears:

```
----------------------------PRINTOUT-SPOOLING-MAIN-MENU---------------------
 OPTION  ===>
                                                       Userid    MBE
                                                       Time      09:15:49



    1    LQ           - List Queue
    2    PO           - Printer Overview
    3    SQ           - Select  Listings
    4    SP           - Select  Printer

   END  EXIT         - Exit Spool
   HELP HELP         - Display Help Information






 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP          END
```

The NSPOOL Main Menu provides access to the major spooling management functions in the middle of the screen.

You can execute the listed functions (LQ, PO, SQ, SP) by either entering the corresponding function number in the OPTION field. You can leave NSPOOL with the END command (**PF3**) or invoke the help system by pressing **PF1**.

The spooling management functions available from the NSPOOL Main Menu are summarized in the following table and discussed in the remainder of this section.

Option	Function	Explanation
1/3	LQ/SQ	Allows spooling display and manipulation specified by printout.
2/4	PO/SP	Allows spooling display and manipulation specified by printer.

Note that the SQ and SP functions allow a more specific selection of output than the LQ and PO functions (see the sections *List Queue* and *Printer Overview*).

General PF Key Assignments

You can use the CLEAR key to return to terminate NSPOOL from any screen. The PF3 key returns you to the previous screen. Note that entering one of these keys on the Main Menu terminates NSPOOL.

Use PF1 to invoke the appropriate help display for the current screen. Type a question mark (?) in any input field to invoke a help window for the field.

List Queue

To display and/or modify one or more printouts, select one of the List Queue functions from the NSPOOL Main Menu.

SQ Function

If you select the SQ function (number 3 on the main menu), a window opens in which you can enter selection criteria to restrict the output to display specific items:

```
------------------------PRINTOUT-SPOOLING-MAIN-MENU------------------------
OPTION  ===> 3

                                                  Userid    MBE
                                                  Time      10:41:07



    1    +-------------------------+
    2    !                         !
    3    ! To confine Display      !
    4    ! of List-Queue,          !
         ! enter Listname: _____ !
   END   !         Format: ____     !
   HELP  !        Printer: _____ !ormation
         !           User: _____ !
         ! Logical Driver: _____ !
         !    Disposition: _        !
         +-------------------------+



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP        END
```

The items in the window correspond to some column headings in the output display (see next figure). The following input values are valid:

name	specific name of the item.
*	select all items (default).
prefix*	select all items with this prefix.
prefix>	display all items, starting from item with this prefix.

SQ/LQ

If you select the function LQ (number 1 from the main menu) or enter selection criteria in the
prompt window invoked by the SQ function, the printout queue is displayed in the following
format:

```
-------------------------------LIST-QUEUE--------------------------------
COMMAND  ===>

  FC   Listname List-No Form Disp Lines Copy Pri   Userid   Log-Drv   Printer
       A>_____          ____ _                   _____  _____ _____
   *   ************************** top of list **************************
   .   RKLLISTE 43            H    199   0    8     RKL                DAEPRT15
   .   HARDCOPY 220           R    50    0    8     JWO                DAEPRT53
   .   HARDCOPY 267           R    50    0    8     TSH                DAEPRT53
   .   HARDCOPY 268           R    36    0    8     JWO                DAEPRT53
   .   RMT      281           R    554   0    8     RMT                STUPRT06
   .   HARDCOPY 283           R    26    0    8     JWO                DAEPRT53
   .   HARDCOPY 289           R    58    0    8     WHE                DUENN535
   .   HARDCOPY 377           R    50    0    8     FMU                DAEPRT63
   .   HARDCOPY 379           R    54    0    8     FMU                DAEPRT63
   .   XCOMV012 473           R    569   0    8     WSL                HAMPRT01
   *   ************************** bottom of list **************************



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP  MENU   END                           DOWN             DATE
```

The above example display was generated by entering *A>* in the*Listname* field of the window in-
voked by the SQ function (see previous figure).

You can modify the output by entering selection criteria in the input fields under some of the
column headings as described under the heading*SQ*above.

Note that each line represents one printout.

If the number of printouts exceeds the screen size, press **PF8** to scroll down.

Press **PF11** to display the date and time the printout(s) were created in place of the Log-Drv and Printer columns. **PF10** redisplays the Log-Drv and Printer names.

The meaning of the fields are explained in the table on the next page.

The following table describes the column headings on the NSPOOL List Queue screen.

Field	Meaning	
FC	Specifies the Function Code. Type the value directly over the "." in the FC column. The following values are permitted:	
	S	SHOW function: Displays the contents of the specified printout at your terminal. (This invokes the "Printout Queue Display".) See the section *Display Printout on Screen*.
	M	MOVE function: Moves the printout to another printer, that is, queues it for another printer and deletes it from the queue of the original printer. Note that this value must be accompanied by a new name in the Printer field. The values for Pri (priority) and Copy (number of copies) can also be modified at the same time.
	C	COPY function: Same as M (above) except that the printout is copied, not moved; that is, it is not deleted from the queue of the original printer.
	P	PURGE function: Purges the printout from the printer queue.
	U	UPDATE function: Updates the specifications of the printout. All fields marked as modifiable can be changed.
Listname	Specifies name of the printout as specified by the originator.	
List-No	Specifies the Com-plete identification number assigned to the printout.	
Form	Specifies the printout form specification (is modifiable).	
Disp	Specifies the disposition of this printout:	
	I	Is inputting, that is, the printout has not finished.
	R	Is ready; is waiting for printer to get ready.
	H	Is ready but will be held until Disp is changed to "R."
	L	Is ready and printed, but is still left in the spooling system.
	O	Is outputting, that is, the printout is currently being printed.
	Note that "R" Disp status can be modified to "H" or a request to leave the printout in the spool (Disp "L") can be made. In addition, "H" can be changed to "R" or a request to leave the printout in the spool ("L") can be made.	
Lines	Specifies the number of text lines for this printout.	
Copy	Specifies the number of additional copies requested for this printout (modifiable).	
Pri	Allows the priority of the printer to be specified. Note that the highest priority is 1. This field is modifiable to generate a list according to priority.	
Userid	Specifies the User ID of the printout originator.	
Either:		

Field	Meaning
Log-Drv	Specifies the name of the logical output driver routine, which can perform additional output formatting during printing. This field is modifiable to generate a list according to logical output driver.
Printer	Specifies the name of the destination for this printout (modifiable with the COPY or MOVE function).
Or:	
Date	Date the printout was created.
Time	Time the printout was created.

Route to System Printer

You can route printouts within the Com-plete TP spooling system to the operating spooling system by using the COPY and MOVE functions (see the preceding table) and defining SYSOUT as the new printer name. The output is then be transferred to the output class "A" as default. To select another output class, specify SYSOUT=x.

List Queue Commands

You can issue any of the following commands from the command line of the List Queue display screen:

Command	Meaning
MENU	Return to NSPOOL main menu.
END	Return to previous screen.
HELP	Display spooling help menu.
PO	Call Printer Overview.
LQ	Call List Queue.
DOWN	Scroll forward.
DATE	Switch display to DATE/TIME mode.
DEST	Switch display to LOG-DRV/PRINTER mode.

The available functions for displayed printouts are described in the above table in the explanation of the column headed FC.

Printer Overview

To display and/or operate one or more printers, select the PO or SP function from the NSPOOL main menu (number 2 or 4 respectively).

SP

If you select the SP function (number 4 on the main menu), a window opens in which you can enter selection criteria to restrict the output to display specific items:

The items in the window correspond to some column headings in the output display (see next figure). The following input values are valid:

name	name of the item (printer name, terminal ID or printer status).
*	select all items (default, not valid for Tid-Nr.).
prefix*	select all items with this prefix (not valid for Tid-Nr.).
prefix>	display all items, starting from item with this prefix (not valid for Tid-Nr.).

Note that the *Printer-Name* field and *Tid-Nr.* field are mutually exclusive.

SP/PO

If you select the function PO (number 2 from the main menu) or enter selection criteria in the prompt window invoked by the SP function, the printer overview is displayed in the following format:

```
-----------------------------PRINTER-OVERVIEW----------------------------
COMMAND  ===>

  FC  Printer      Tid  Dev-Type   Status     Form   Q-Num

      _____     ____            _____
  *   **************************** top of list ****************************
  .   SYSOUT       1    BATCH      WAIT                0
  .   DUPPRT14     2    3288  L    SIMLOGO             3
  .   DAEPTR75     3    3288  L    SIMLOGO             2
  .   SAEPRT25     4    3288  L    SIMLOGO             1
  .   DAERPT35     5    3288  L    SIMLOGO             2
  .   DUGPR14      6    3288  L    SIMLOGO             1
  .   HUGO         7    3288  L    SIMLOGO             2
  .   LPRTSM02     8    3288  L    SIMLOGO             1
  .   LPRTSM03     9    3288  L    SIMLOGO             1
  .   LPRTSM04     10   3288  L    SIMLOGO             1
  .   FF           11   3288  L    SIMLOGO             2
  .   DAEPRT1      12   3288  L    SIMLOGO             2
  .   MZCPRT       13   3288  L    SIMLOGO             3
  .   XXX00001     14   3288  L    SIMLOGO             1
  .   DAEPTR30     15   3288  L    SIMLOGO             3
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP  MENU  END                   TOP        DOWN
```

The above example display was obtained by invoking the SP function and leaving all fields blank in the prompt window (see previous figure).

You can modify the output by entering selection criteria in the input fields under some of the column headings as described under the heading SP above.

Note that each printer name can be associated with either a real or virtual printer.

If the list of printers exceeds the screen size, press **PF8** to scroll down. Press **PF6** to redisplay the first output page.

The meaning of the fields according to column heading is described in the table on the next page:

The following table describes the column headings on the NSPOOL Printer Overview screen.

Field	Meaning
FC	Specifies the function code. Type the value directly over '.' in the FC column. The following values are permitted:
M	MOUNT new form function: Mount another output format on the printer. Note that this value must be accompanied by a new entry in the "FORM" field.
S	START printer function: start a printer that has been halted.
R	RESET function: recover after I/O error.

Field	Meaning
H	HALT function: halt current printout.
P	POSITION printout function: positions the current printout, that is, stops printing and resumes printing at any specified page within the printout. You specify the place at which printing is to be resumed by filling the Mode and Num parameters in the prompt window that appears when invoking the P function. Possible values for the Mode field are:
	<table><tr><td>T</td><td>Top</td><td>Printing will resume from the top of the printout.</td></tr><tr><td>B</td><td>Bottom</td><td>Printing will resume from the bottom of the print-out (last page is printed).</td></tr><tr><td>A</td><td>Absolute</td><td>Printing will resume at the page specified in the Num field.</td></tr><tr><td>R</td><td>Relative</td><td>Printing will resume nnn pages from the current page forward or backward. The number of pages must be specified in the Num field. Forward pages are specified by the number, backward pages the number preceded by the minus sign(e.g., -5).</td></tr></table>
Q	QUEUE function: Displays the queue for the printer, that is, switches to the List Queue of Printouts display. (This function is the same as selecting the NSPOOL Main Menu LQ function, described earlier in this section).
C	CANCEL function: Cancels the current printout for the printer, that is, the List Queue of the current printout is displayed. The field FC is filled with value "P". Press to purge the list. Printing will resume with the next printout.
F	FLUSH function: Cancels all queue entries for this printer. Is the same as cancel current printout for all printouts of the printer.
Printer	Specifies the logical name of the printer.
Tid	Specifies the unique Terminal ID as specified in the TIBTAB.
Dev-Typ	Specifies the device type of the printer.
Status	Specifies the current status of the printer. Possible values:
	<table><tr><td>WAIT</td><td>Is waiting for work.</td></tr><tr><td>RUN</td><td>Is currently active.</td></tr><tr><td>HALT</td><td>Is held up due to operator intervention.</td></tr><tr><td>ERROR</td><td>Is held due to physical I/O error during output.</td></tr><tr><td>SIMLOGO</td><td>Is currently in simlogon required status. This status is indicated if output was scheduled for this printer and Com-plete is waiting for a session to be established.</td></tr><tr><td>INTVREQ</td><td>Is waiting for operator intervention (for example, paper is out), before the printer can be put online.</td></tr><tr><td>UNDEFIN</td><td>Is currently undefined in the spooling system.This status is indicated if output was scheduled for this printout, but Com-plete has no corresponding active session.</td></tr></table>
Form	Specifies the output form currently mounted on printer. You can change the form by typing M in the function field, a new form ID in the Form field and pressing .
Q-Num	Number of printouts in the queue for this printer.

Printer Overview Commands

You can issue any of the following commands from the command line of the Printer Overview display screen:

Command	Meaning
MENU	Return to NSPOOL main menu.
END	Return to previous screen.
HELP	Display spooling help menu.
PO	Call Printer Overview.
LQ	Call List Queue.
DOWN	Scroll forward.
TOP	Restart printer overview from the top.

The available functions for displayed printers are described in the above table in the explanation of the column headed FC.

NSPOOL Display Printout on Screen (SHOW or QUEUE Function)

The SHOW or QUEUE function enables you to display the contents of a specific printout at your terminal. You can request the display from either the List Queue screen (Function Code "S") or from the Printer Overview screen (Function Code "Q").

The following figure shows a sample of the output produced by selecting this function.

```
------------------------------PRINTOUT-DISPLAY------------------------------
PRINTOUT: RKLLISTE                                          COLUMNS 001 079
COMMAND  ===>

 ******************************* top of list *******************************
 109:03:50        Current Object  MTMENUOP in library SG--PROD        92-02-05
  0010 * MTMENUOP SAG System Products Maintenance Tool Menu        GW 88-02-06
  0020 *
  0030 DEFINE DATA
  0040    GLOBAL USING MT00000G
  0050    LOCAL  USING +PFK-LDA
  0060    LOCAL  USING MTMENUOL
  0070    LOCAL  1 #I (I2)
  0080 END-DEFINE
  0090 *
  0100 * ----------------------------------------------------------------------
  0110 DEFINE SUBROUTINE INITIALIZATION
  0120 * ----------------------------------------------------------------------
  0130 *
  0140 ASSIGN #PF-VALUE (3)  = 'END'
  0150 ASSIGN #PF-VALUE (6)  = 'SYS'
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
                   END                              DOWN      LEFT  RIGHT
```

When viewing the printout, use **PF8** to display the next page (if there are more lines available). To shift the start column of the output to the left, use **PF10** to shift to the right, use **PF11**. The maximum line size of the output is 256 characters. Normally the columns 1 to 79 will be shown. Most of the printouts have 133 columns. In this case, a shift right shows column 55 to 133, and a shift left 1 to 79. If more columns available, it will be shifted to the next 79 lines to the right or left. To return to previous screen, press **PF3**.

Printout Display Commands

You can issue any of the following commands from the command line of the printout display screen:

Command	Meaning
END	Return to previous screen.
HELP	Display spooling help menu.
PO	Call Printer Overview.
LQ	Call List Queue.
DOWN	Scroll forward.
LEFT	Shift display to the left.
RIGHT	Shift display to the right.

Customization

NSPOOL functionality is provided by the module CSPOOL which is provided in source format (see the section Installation Considerations above).

CSPOOL references the CMSPCB macro. This macro generates the DSECT for the control block SPCB, as well as the DSECTs for the input and output lines (inp-arg, out-arg), depending on the selected function. For a description of the macro, see the CMSPCB copybook on the supplied source library.

The NSPOOL user interface is written in Natural. The Natural front-end programs use the data areas for communication with CSPOOL listed in the following table:

Source	Structure	Length	Remarks		
SPWK01-A	#SPWK01A OP	4	For INIT/TERM		
SPCB01-A	#SPCB SPCB	200			
SPQI01-A	#QI-WRKLINE inp-arg	80			
SPQO01-A	#QO-WRKLINE out-arg	200	All subfunctions except DP and DPN		
SPQO02-A	#DPO-WRKARRAY out-arg	256	Subfunctions DP and DPN only		

Parameter Areas

SPWK01-A

```
*    TITLE........: Common Work-area for Communication with SPCB01-N
*    NAME.........: SPWK01-A
*    TYPE.........: A
* - ------------------------------ - ---- ----------------------
  1 #SPWK01A
  2 #OP                          A    4
R 2 #OP                                   /* REDEF. BEGIN : #OP
  3 #OP-1                        A    1
  3 #OP-2                        A    1
  3 #OP-3                        A    1
  3 #OP-4                        A    1
```

SPCB01-A

```
*    TITLE........: Communication-area for corresponding via CSPOOL
*    NAME.........: SPCB01-A
*    TYPE.........: A
* - ------------------------------ - ---- ----------------------
  1 #SPCB                          A  200
R 1 #SPCB                                   /* REDEF. BEGIN : #SPCB
  2 #SP-ID                         B    4 /* 'SPCB'
  2 #SP-DATO                       B    4 /* INTERNAL
  2 #SP-DATI                       B    4 /* INTERNAL
  2 #SP-REQID                      B    4
  2 #SP-REQIDC                     B    4
  2 #SP-OP                         A    4
  2 #SP-LREQ                       B    2
  2 #SP-LLEN                       B    2
  2 #SP-DISP                       B    2 /* DISPL. IN LINE
  2 #SP-RETCD                      B    2
  2 #SP-MSGNUM                     B    2
  2 #SP-RLINS                      B    2
  2 #SP-DYNW                       B    4 /* INTERNAL
  2 #SP-DYNWL                      B    4 /* INTERNAL
  2 FILL1                          A   28
  2 #SP-LNAMEL                     B    2
  2 #SP-LNAME                      A    8
  2 FILL2                          A    8
  2 #SP-FORML                      B    2
  2 #SP-FORM                       A    8
  2 FILL3                          A    8
  2 #SP-DESTL                      B    2
  2 #SP-DEST                       A    8
  2 FILL4                          A    8
  2 #SP-USERL                      B    2
  2 #SP-USER                       A    8
  2 FILL5                          A    8
  2 #SP-LDRVL                      B    2
  2 #SP-LDRV                       A    8
  2 FILL6                          A    8
  2 #SP-STATL                      B    2
  2 #SP-STAT                       A    8
  2 FILL7                          A   28
```

SPQI01-A

```
*   TITLE........: input  Work-line
*   NAME.........: SPQI01-A
*   TYPE.........: A
* - ----------------------------- - ---- --------------------------
  1 #QI-WRKLINE                     A   80
  1 #QI-WRKLINE                                /* RED. BEGIN : #QI-WRKLINE
  2 #LQI-ID                         B    4
  2 #LQI-NAME                       A    8
  2 #LQI-NUM                        B    2
  2 #LQI-FORM                       A    4
  2 #LQI-STAT                       A    1
  2 #LQI-LINS                       B    2
  2 #LQI-COPIES                     I    2
  2 #LQI-PRIO                       I    2
  2 #LQI-USER                       A    8
  2 #LQI-LDRV                       A    8
  2 #LQI-DEST                       A    8
  2 #LQI-DATE                       A    8
  2 #LQI-TIME                       A    8
  2 #LQI-FILL1                      A   15
R 1 #QI-WRKLINE                                /* RED. BEGIN : #QI-WRKLINE
  2 #PQI-ID                         B    4
  2 #PQI-NAME                       A    8
  2 #PQI-TID                        B    2
  2 #PQI-DTYPE                      A    8
  2 #PQI-FORM                       A    4
  2 #PQI-NUM                        B    2
  2 #PQI-STAT                       A    8
  2 #PQI-FILL1                      A   44
```

SPQO01-A

```
*   TITLE........: output Work-line
*   NAME.........: SPQO01-A
*   TYPE.........: A
* - ----------------------------- - ---- --------------------------
  1 #QO-WRKLINE                     A  200
R 1 #QO-WRKLINE                                /* RED. BEGIN : #QI-WRKLINE
  2 #LQO-ID                         B    4
  2 #LQO-NAME                       A    8
  2 #LQO-NUM                        B    2
  2 #LQO-FORM                       A    4
  2 #LQO-STAT                       A    1
  2 #LQO-LINS                       B    2
  2 #LQO-COPIES                     I    2
  2 #LQO-PRIO                       I    2
  2 #LQO-USER                       A    8
```

```
  2 #LQO-LDRV                       A     8
  2 #LQO-DEST                       A     8
  2 #LQO-DATE                       A     8
  2 #LQO-TIME                       A     8
  2 #LQO-FILL1                      A   135
R 1 #QO-WRKLINE                                /* RED. BEGIN : #QI-WRKLINE
  2 #PQO-ID                         B     4
  2 #PQO-NAME                       A     8
  2 #PQO-TID                        B     2
  2 #PQO-DTYPE                      A     8
  2 #PQO-FORM                       A     4
  2 #PQO-NUM                        B     2
  2 #PQO-STAT                       A     8
  2 #PQO-FILL1                      A   164
```

SPQO02-A

```
*    TITLE........: output Work-line only for display Printout
*    NAME.........: SPQO002-A
*    TYPE.........: A
* - ------------------------------ - ---- ---------------------------
  1 #DPO-WRKARRAY
  2 #DPO-WRKLINE                    A   200
R 2 #DPO-WRKLINE                                /* RED. BEGIN: #DPO-WRKLINE
  3 #DPO-ID                         B     4
  3 #DPO-NAME                       A     8
  3 #DPO-NUM                        B     2
  3 #DPO-FORM                       A     4
  3 #DPO-STAT                       A     1
  3 #DPO-LINS                       B     2
  3 #DPO-COPIES                     I     2
  3 #DPO-PRIO                       I     2
  3 #DPO-USER                       A     8
  3 #DPO-LDRV                       A     8
  3 #DPO-DEST                       A     8
  3 #DPO-DATE                       A     8
  3 #DPO-TIME                       A     8
  3 #DPO-FILL1                      A   135
  2 #DPO-A56                        A    56
```

Supported Functions and Subfunctions

The following table provides an overview of the functions and subfunctions supported by NSPOOL's Natural front-end.

Function	Subfunction	Meaning
PO		List printer overview
	OP	Operate printer
	PP	Modify printer
LQ		List printout queue
	DP	Display printout
	MP	Modify printout

General Programming Considerations

When customizing the NSPOOL front-end, the following points must be considered:

Logic in general:

```
   CALL 'CSPOOL' 'INIT' SPCB
// CALL 'CSPOOL' SPCB inp-arg out-arg //
   CALL 'CSPOOL' 'TERM' SPCB
```

Initialization

The storage areas for the SPCB must be provided by the calling program. It is initialized by the following call:

```
CALL 'CSPOOL' 'INIT' SPCB
```

The reserved fields in the SPCB must not be modified by the user program. CSPOOL will keep addresses of internal work areas in those fields.

Termination

To release the storage areas acquired by CSPOOL, a termination call is required:

```
CALL 'CSPOOL' 'TERM' SPCB
```

For more detailed information of CALL 'CSPOOL' ('INIT' / 'TERM') SPCB see frontend program SPCB01-N.

Communication

The calling program sets the function code in the field #SP-OP in the #SPCB(SPCB-name in the frontend programs). The keyword table in the #SPCB (#SP-NAMEL #SP-NAME etc.) can be filled to specify search criterias. Call :

```
CALL 'CSPOOL' #SPCB #QI-WRKLINE #DPO-WRKARRAY (Names in frontend
CALL 'CSPOOL' #SPCB #QI-WRKLINE #QO-WRKLINE    programs)
```

Note that *inp-arg* points to a copy of the *out-arg* of a previous (LQ/PO) request, if any subfunction is used. The field #SP-REQID in the #SPCB must also contain the value from (#LQO-ID/#PQO-ID).

Printer Overview

Function	Meaning
PO	first entry
POS	same entry again / position to referred entry
PON	next in queue

The following table lists the operands that must be filled in the used front-end program before calling CSPOOL. Which operands are relevant to which functions can be seen from the matrix following the table.

Data area	Fields in #SPCB	Format/length	Value
SPCB01-A	#SP-OP	A4	Operation code (PO, POS, PON)
SPCB01-A	#SP-LREQ	B2	1
SPCB01-A	#SP-REQID	B4	TIB address
SPCB01-A	#SP-DESTL	B2	Length of destination name or TID number
SPCB01-A	#SP-DEST	A8	Destination name or TID number

Matrix:

-	PO, POS	PO, POS	PO, POS, PON
#SP-OP	(1) in	(1) in	(1) in
#SP-LREQ	(1) in	(1) in	(1) in
#SP-REQID	(2) in	-	-
#SP-DESTL	-	(3) in	(3) in
#SP-DEST	-	(2) in	(2) in

Key:

(1)	Required.
(2)	Optional.
(3)	If destination-name (#SP-DEST) is filled, the length of destination-name is required in #SP-DESTL.
in	Input field

For more detailed information, see the source of front-end program SPPQ01-S.

Operate Printer

Operate Printer (OP) is a subfunction of the Printer Overview function and consists of the following:

OPM	mount printer format
OPS	start printer
OPR	reset printer
OPH	halt printer

The following table lists the operands that must be filled in the used front-end program before calling CSPOOL. Which operands are relevant to which functions can be seen from the matrix following the table.

Data area	Fields in #SPCB	Format/length	Value
SPCB01-A	#SP-OP	A4	Operation code (OPM, OPS, OPR, OPH)
SPCB01-A	#SP-LREQ	B2	1
SPCB01-A	#SP-REQID	B4	TIB address (#PQD-ID)
SPQI01-A	#QI-WRKLINE	A80	Out-arg (#QO-WRKLINE)
SPCI01-A	#PQI-FORM	A4	Printer format

Matrix:

	OPM	OPS	OPR	OPN
#SP-OP	(1) in	(1) in	(1) in	(1) in
#SP-LREQ	(1) in	(1) in	(1) in	(1) in
#SP-REQID	(1) in	(1) in	(1) in	(1) in
#QI-WRKLINE	(1) in	(1) in	(1) in	(1) in
#PQI-FORM	(1) in	-	-	-

Key:

(1)	Required.
in	Input field

Subfunction OP is only possible after a successful function PO. For more detailed information, see the source of front-end program SPOP01-P.

Position Current Printout

This subfunction branches to the function List Queue Overview. For execution, #SP-DESTL and #SP-DEST are filled. Subfunction Position Current Printout can then be used as described below.

Position Current Printout is a subfunction of List Queue Overview and consists of the following:

PPT	position current printout top
PPB	position current printout bottom
PPA	position current printout absolute
PPR	position current printout relative

The following table lists the operands that must be filled in the used front-end program before calling CSPOOL. Which operands are relevant to which functions can be seen from the matrix following the table.

Data area	Fields in #SPCB	Format/length	Value
SPCB01-A	#SP-OP	A4	Operation code (PPT, PPB, PPA, PPR)
SPCB01-A	#SP-LREQ	B2	With PPA and PPR, number of pages requested. Otherwise, 1
SPCB01-A	#SP-REQID	B4	TIB address (#PQD-ID)
SPQI01-A	#QI-WRKLINE	A80	Out-arg (#QO-WRKLINE)

Matrix:

	PPT	PPB	PPA	PPR
#SP-OP	(1) in	(1) in	(1) in	(1) in
#SP-LREQ	(1) in	(1) in	(1) in	(1) in
#SP-REQID	(1) in	(1) in	(1) in	(1) in
#QI-WRKLINE	(1) in	(1) in	(1) in	(1) in

Key:

(1)	Required.
in	Input field

Subfunction PP is only possible after a successful PO and LQ request. For more detailed information, see source of front-end program SPPP01-S.

EXCEPTION!

The following two subfunctions are simulated.

OPC	cancel current printout
OPF	flush all queue entries

With these subfunctions, the function List Queue Overview is branched to. For execution, #SP-DESTL and #SP-DEST are filled. Subfunction Modify Queue entry purge (MQP) can then be used as described below.

List Queue Overview

Function	Meaning
LQ	first entry
LQS	same entry again / position to referred entry
LQN	next in queue

The following table lists the operands that must be filled in the used front-end program before calling CSPOOL. Which operands are relevant to which functions can be seen from the matrix following the table.

Data area	Fields in #SPCB	Format/length	Value
SPCB01-A	#SP-OP	A4	Operation code (LQ, LQS, LQN)
SPCB01-A	#SP-LREQ	B2	1
SPCB01-A	#SP-REQID	B4	MCQ address
SPCB01-A	#SP-LNAMEL	B2	Length of printout name
SPCB01-A	#SP-LNAME	A8	Printout name
SPCB01-A	#SP-FORML	B2	Length of format name
SPCB01-A	#SP-FORM	A8	Format name
SPCB01-A	#SP-DESTL	B2	Length of destination name
SPCB01-A	#SP-DEST	A8	Destination name
SPCB01-A	#SP-USERL	B2	Length of user ID

Data area	Fields in #SPCB	Format/length	Value
SPCB01-A	#SP-USER	A8	User ID
SPCB01-A	#SP-LDRVL	B2	Length of logical driver name
SPCB01-A	#SP-LDRV	A8	Logical driver name
SPCB01-A	#SP-STATL	B2	Length of status name
SPCB01-A	#SP-STAT	A8	Status name

Matrix:

	LQ, LQS	LQ, LQS, LQN	LQ, LQS, LQN
#SP-OP	(1) in	(1) in	(1) in
#SP-LREQ	(1) in	(1) in	(1) in
#SP-REQID	(2) in	-	-
#SP-LNAMEL	-	(3) in	-
#SP-LNAMEL	-	(2) in	-
#SP-FORML	-	(3) in	-
#SP-FORM	-	(2) in	-
#SP-DESTL	-	(5) in	-
#SP-DEST	-	(2) in	-
#SP-USERL	-	(6) in	-
#SP-USER	-	(2) in	-
#SP-LDRVL	-	(7) in	-
#SP-LDRV	-	(2) in	-
#SP-STATL	-	(8) in	-
#SP-STAT	-	(2) in	-

Key:

in	Input field
(1)	Required.
(2)	Optional.
(3)	If printout-name (#SP-LNAME) is filled, the length of the printout name is required in #SP-LNAMEL.
(4)	If format-name (#SP-FORM) is filled, the length of the format name is required in #SP-FORML.
(5)	If destination-name (#SP-DEST) is filled, the length of the destination name is required in #SP-DESTL.
(6)	If user-id (#SP-USER) is filled, the length of the user ID is required in #SP-USERL.
(7)	If logical driver-name (#SP-LDRV) is filled, the length of the logical driver-name is required in #SP-LDRVL.
(8)	If status-name (#SP-STAT) is filled, the length of the status name is required in #SP-STATL.

For more detailed information, see source of frontend program SPLQ01-N.

Printout Display

Printout Display is a subfunction of List Queue Overview and consists of the following:

DP	first entry
DPN	next line

The following table lists the operands that must be filled in the used front-end program before calling CSPOOL. Which operands are relevant to which functions can be seen from the matrix following the table.

Data area	Fields in #SPCB	Format/length	Value
SPCB01-A	#SP-OP	A4	Operation code (DP, DPN)
SPCB01-A	#SP-LREQ	B2	1
SPCB01-A	#SP-REQID	B4	MCQ address (#LQO-ID)
SPCB01-A	#SP-LLEN	B2	Length of output line (max. 256)
SPQI01-A	#QI-WRKLINE	A80	Out-arg (#QO-WRKLINE)

Matrix:

-	DP	DPN
#SP-OP	(1) in	(1) in
#SP-LREQ	(1) in	(1) in
#SP-REQID	(1) in	-
#SP-LLEN	(1) in	-
#QI-WRKLINE	(1) in	-

Key:

in	Input field
(1)	Required.

Subfunction DP is only possible after a successful LQ request. For more detailed information, see source of front-end program SPDP01-P.

Modify Queue Entry

Modify Queue Entry is a subfunction of List Queue Overview and consists of the following:

MQU	modify queue entry update
MQP	purge queue entry
MQC	copy queue entry
MQM	move queue entry

The following table lists the operands that must be filled in the used front-end program before calling CSPOOL. Which operands are relevant to which functions can be seen from the matrix following the table.

Data area	Fields in #SPCB	Format/length	Value
SPCB01-A	#SP-OP	A4	Operation code (MQU, MQP, MQC, MQM)
SPCB01-A	#SP-LREQ	B2	1
SPCB01-A	#SP-REQID	B4	MCQ address (#LQO-ID)
SPQI01-A	#QI-WRKLINE	A80	Out-arg (#QO-WRKLINE)
SPQI01-A	#LQI-FORM	A4	Printout format
SPQI01-A	#LQI-STAT	A1	Printout status
SPQI01-A	#LQI-COPIES	B2	Number of printout copies
SPQI01-A	#LQI-PRIO	B2	Printout priority
SPQI01-A	#LQI-LDRV	A8	Printout logical driver
SPQI01-A	#LQI-DEST	A8	Printout destination

Matrix:

-	MQU	MQP	MQC	MQM
#SP-OP	(1) in	(1) in	(1) in	(1) in
#SP-LREQ	(1) in	(1) in	(1) in	(1) in
#SP-REQID	(1) in	(1) in	(1) in	(1) in
#QI-WRKLINE	(1) in	(1) in	(1) in	(1) in
#LQI-FORM	(2) in	-	(2) in	(2) in
#LQI-STAT	(2) in	-	(2) in	(2) in
#LQI-COPIES	(2) in	-	(2) in	(2) in
#LQI-PRIO	(2) in	-	(2) in	(2) in
#LQI-LDRV	(2) in	-	(2) in	(2) in
#LQI-DEST	-	-	(1) in	(1) in

Key:

(1)	Required.
(2)	Optional.
in	Input field.

Subfunction MQ is only possible after a successful LQ request. For more detailed information, see source of front-end program SPMQ01-P.

V Miscellaneous Functions

This part of the Application Programming documentation describes some miscellaneous API functions which enable the application program to utilize a wide range of features.

The functions available are described in the following table. All functions can be used in the same application program, either repeatedly or singly.

The information is organized under the following headings:

Function	Description
ABEND Function	Abnormally terminate with a specific termination code at a specific location within the program. A dump entry is automatically created in the Com-plete online dump file, thus enhancing program testing by providing a program image at the time of termination.
ABEXIT Function	Set up routine to get control when a program abends.
CMPOST Function	Signal completion of an event.
CMWAIT Function	Wait on up to to eight events.
COMSTOR Functions	Obtain, modify, and release storage outside of the application thread area.
DATE Function	Obtain the current date in either Julian or Gregorian format.
EOJ Function	Normally terminate an application program.
FREEMAIN Function	Free storage in a thread.
GETCHR Function	Interrogate information pertaining to the terminal in use and/or the terminal user. This information can subsequently be used to establish security for the application program or to determine various logic paths to be followed based upon desired criteria.
GETMAIN Function	Acquire storage in a thread.
GETSTOR Function	Acquire storage in a thread, above the 16 MB line, if available.
LEPIPI Function	Call LE program from non-LE program.
MODIFY Function	Modify the program environment.
RJE Function	Submit job streams to the batch environment for scheduling of execution.
ROLEVT Function	Rollout of the thread until an event controlled by a companion JOB (or task) occurs.

Function	Description
ROLOUT Function	Force a scheduled rollout operation for the thread in which it is executing. The rollout operation can be used to free the thread resource.
SETEID Function	Recognize the entry of the desired Program Attention keys or Program Function keys. Since Com-plete normally recognizes entry of the keys, the application program will normally not be notified of the use. The SETEID function is used to mask off from Com-plete the desired Program Attention or Program Function keys.
SNAP Function	Write a thread dump without abending.
TESTAT Function	Test for a desired program interrupt condition caused by pressing a Program Attention key. This function is normally used in conjunction with the ROLOUT function or the WRT function in order to enable the application program to terminate a timed rollout operation.
TIME Function	Obtain information about timing intervals.

11 ABEND Function

The ABEND function initiates Com-plete abnormal termination processing for an application program. A hex dump of the program storage area is produced. The program does not regain control.

An abend code can be specified by using the optional *abcode*argument. This code is displayed at the terminal for online programs, and is included in the printout for batch programs. The largest number that can be given for the abend code is 9999. If the number is omitted or is larger than 9999, Com-plete will initialize the abend code to 255.

The ABEND function is useful when debugging an application program. Several ABEND CALL statements can be coded, each with an *abcode* value, to enable tracing program logic by halting execution at various node points. The *abcode* returned to the screen or printout data set facilitates the following of the execution logic of the program, and the dump-generated aids in identifying and correcting errors.

Format

The format for using the ABEND function is:

```
ABEND [(abcode)]
```

abcode	Optional. Default: 255 The user-supplied abend termination code. The largest value permitted is 9999. If the abcode is omitted or is larger than 9999, Com-plete defaults to 255.

Return Codes

No return codes are provided by the ABEND function because control is not returned to the application program.

12 ABEXIT Function

📄 **Note:** This function can only be used with programs written in Assembler.

The ABEXIT function enables an application program to set up a routine which will get control in the event of the program abending. This can be used to clean up reserved resources, to provide an application specific message to users of the application and/or to enable the application to continue processing if desired.

The exit routine will receive control in the same context as it was set up. This means that the registers are saved at the point where the exit is set up and the user exit given control with these same registers if an abend occurs. If the ABEXIT function was issued in 31 bit mode, or the exit is a 31 bit address, the exit will get control in 31 bit mode. It will get control in the key in which the thread is in at the time of the abend.

If another abend occurs while the exit has control, the exit will not get control a second time and the application program will be abended. An exit is deemed to be in control until the point that it successfully issues another API request.

An abend exit can also be used as an end-of-job exit routine. If you specify the EOJ parameter on the ABEXIT function call, then this abend exit routine will also receive control when Com-plete terminates the application normally. To distinguish between an abend call and an end-of-job call, the exit routine can analyze the contents field ABXDCODE (abend code), which will contain zero at an end-of-job call.

The format for the ABEXIT function is:

```
ABEXIT ( retcode , exit address , ABXD [,eoj] )
```

retcode	A fullword containing the return code after Com-plete has processed the appropriate function.
exit address	This is the address of the exit routine to get control in the event of an abend. If this field is zero, any previously set abend exit routine will be reset.
ABXD	This is the address of the user supplied abend exit data area as mapped by the copybook CCABXD. This parameter is required if the request is to set an abend exit. Refer to the copybook itself for details of the contents which are filled out when an abend occurs. This parameter must be supplied for both set and reset requests.

Return Codes

0	Abend exit set successfully set or reset
4	Exit replaced a previously set exit for a set request. No previous exit set for a reset request
8	Insufficient thread storage to process request
12	Invalid parameters passed to the ABEXIT function

Abends

Abends may occur if the exit address and/or the ABXD address are invalid.

13 CMPOST Function

The CMPOST function can be used to signal that an event has completed. The event in question is indicated via an Event Control Block (ECB) which must previously have been or is about to be the object of a CMWAIT or ROLEVT request for the application program waiting on the event to complete.

A program catalogued PV may post an ECB anywhere in the Com-plete address space. A program catalogued non PV may only post an ECB which is within a previously allocated COMSTOR area.

The format for the CMPOST function is:

```
CMPOST ( Return code , ECB , post code )
```

retcode	A fullword containing the return code after Com-plete has processed the appropriate function.
ECB	This is the address of the Event Control Block (ECB) to be posted active. Note that for the post to have any effect, another application must be waiting on the ECB in question.
post code	This is the address of an optional fullword, the contents of which will be placed in the ECB when it has been posted active. This can be useful to determine what posted the ECB active if it can be posted from different parts of the system.

Return Codes

0	ECB posted successfully
4	Invalid parameter list supplied. The ECB in the list does not exist or is not fullword aligned
8	No COMSTOR available
12	ECB provided by non PV user was not in a COMSTOR area

Abends

There are no abends expected with the normal usage of this function.

14 CMWAIT Function

The CMWAIT function may be used to wait on up to a maximum of eight events. This call provides the Com-plete dispatching nucleus with the ability to service other users while the current application waits for what should normally be a short term event. As the program cannot be rolled out over this call, the thread will be unavailable to other users. Therefore, if it is expected that the event will take longer, the ROLEVT function should be used instead.

The format for the CMWAIT function is:

```
CMWAIT ( retcode , ECB1 , ECB2 ......., ECB8 )
```

retcode	A fullword containing the return code after Com-plete has processed the appropriate function.
ECB1 to ECB8	The address of the Event Control Block(s) upon which the program would like to wait. Programs which are catalogued PV may wait on up to 8 ECBs which must exist somewhere in the Com-plete address space. Programs which are catalogued non PV may only wait on 1 ECB and this ECB must be within a previously allocated COMSTOR area. Only one ECB may exist in any one COMSTOR area.

Return Codes

0	One or more of the events has been posted complete
4	Invalid parameter list supplied One of the ECBs in the list does not exist or is not fullword aligned The same ECB address was supplied twice in the list. For OS systems, one of the ECBs had the wait bit on. A non PV user supplied more than one ECB for the request.
8	No COMSTOR available.
12	ECB provided by non PV user was not in a COMSTOR area.
16	The ECB in COMSTOR is already being waited upon

Abends

There are no abends expected with the normal usage of this function.

15 **COMSTOR Functions**

The Common Storage Control (CSC) functions provide a facility that allows the application program to obtain, modify, and release storage outside of the application thread area. This facility allows a Com-plete application program to interact with another application program by sharing "common" storage. The storage is not located within the thread, but is located outside the thread in an area that is managed by Com-plete.

Com-plete's CSC routines use a control block to communicate with the application program. Information about the request, such as function type (for example, GEN, GET), is placed in the control block before invoking the COMSTOR routine. Com-plete returns the status of the call and other information to the application program in the control block.

Several function types are available to the application program to maintain this storage. They include:

- GEN - obtains and initializes storage;
- FREE - frees the storage;
- PUT - modifies part or all of the storage by moving data into it from the application program;
- GET - "reads" the storage by moving part or all of it into the application program.

GEN is used to obtain working storage outside of the thread. Specifically, the storage is not rolled out with the thread when the application program is rolled out; however, the storage is available to other application programs while the program that obtained and initialized it is rolled out. The storage is obtained from a special buffer pool managed by Com-plete and cannot be modified directly.

The *area* parameter is used to initialize the storage obtained by the GEN call. If no *area* parameter is supplied, the area is not initialized.

Note that the storage obtained by a COMSTOR GEN function call is considered "non-accountable", that is, the storage is controlled by the application and can only be freed by an explicit call from an application program. Synchronization of the access to the storage is left up to the application programs involved.

FREE is used to free the storage previously obtained by GEN. After control is returned from the FREE function call, the storage is no longer available to any application programs. The storage ID (see the CSCID field explained in the table below) can be re-used to obtain and initialize a new piece of storage.

PUT is used to modify the storage that was obtained with GEN. The entire area, or just a part of it, can be modified. The *controlblock* parameter specifies the offset and length of the data in the GENed area that is to be modified. The *area* parameter is the name of a field containing data to be used for the modification.

GET is used to "read" the storage previously obtained with GEN and/or modified by PUT. The entire GENed area, or just a part of it, can be moved into the application. The *controlblock* parameter

specifies the offset and length of the data in the GENed area to be moved into the application program. The *area* parameter is the name of a field that is the target for the data to be moved.

Format

The format for using the COMSTOR function is:

```
COMSTOR (retcode, controlblock , area)
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
control-block	Required. The label of a data area containing the common storage control block. Refer to the CSC control block table below. The desired function is indicated within this control block.
area	The label of a data area, the usage of which is dependent upon the specified function.Optional.

Return Codes

After each call, the return code must be examined. All return codes are posted in the first parameter, as follows:

0	The function completed normally.
4	The function did not complete. The control block address is invalid.
8	The function did not complete. Refer to the CSCFBK field in the control block for the feedback code.

Feedback Codes

If Com-plete finds the location of the CSC control block to be valid but discovers some other error, the application program receives return code 8 and the CSCFBK field contains one of the following feedback codes:

4	The storage already exists (GEN), or the storage does not exist (FREE, PUT, GET).
8	The amount of storage requested is not available (GEN).
12	Control storage is unavailable. Too many GEN requests are outstanding. Notify the system programmer.
16	INVALID function.
20	AREA PARM missing.
24	AREA length (CSCLEN) invalid.
28	DATA offset (CSCOFF) invalid.
32	The storage is held by ROLEVT and cannot be changed (FREE, PUT).

Abends

There are no abends associated with the Common Storage Control functions.

CSC Control Block

Com-plete Common Storage Control uses a control block to pass information back and forth between itself and the application program. The control block is the second parameter in each call to the COMSTOR function, and is formatted as shown in the following table:

Name	Length	Offset	Description
CSCNAME	4	0	Character; must be CSCB
CSCFUNC	4	4	Character; function - must be GEN, GET, PUT, or FREE
CSCID	8	8	Character; storage identifier - used to uniquely identify (i.e., "name") the storage
CSCLEN	4	16	Signed binary; data length - the length of the data to be moved to or from the GENed area.
CSCOFF	4	20	Signed binary; offset to data - the offset in the GENed area to which, or from which, the data is to be moved.
CSCAREA	4	24	Storage address; address of the GENed area in Com-plete storage. The storage pointed to by this address should not be directly referenced; it is in protected storage. If an ECB is built as the first word of the GENed area, the data name of CSCAREA can be used as the ECB parameter of a Com-plete ROLEVT function.
CSCRC	2	28	Signed binary; return code.
CSCFBK	2	30	Signed binary; feedback code - indicates the status of the COMSTOR call. See feedback codes in previous section for valid values.

16 DATE Function

The DATE function enables the application to obtain one of the following:

- Gregorian date: mmddyy
- Julian date: yyddd

The current date can optionally be obtained using the standard COBOL reserved word CURRENT-DATE.

DATE indicates that the date is to be returned in Gregorian format.

DATEJ indicates that the date is to be returned in Julian format.

Format

The format for using the DATE function is:

```
DATE[J] (retcode, area)
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
area	Required. A four-byte area in the working storage area of the application program where the date will be placed.

The date can be returned in either Gregorian or Julian format. Regardless of the format chosen, the date is returned in a four-byte area identified by the area argument. The format of the data returned in the four-byte field is:

Gregorian Format: *X′cmmddyyF′*

The first character contains either a leading half-byte of zeros, or the century indicator "c". The characters are:

c = century indicator (c=0=1900; c=1=2000; etc.)
mm = month indicator (01 to 12)
dd = date indicator (01 to 31)
yy = year indicator (00 to 99)

Note that the Gregorian format illustrated above differs from the results returned by the MCALL DATE function for Assembler language programming.

Julian Format: *X'0cyydddF'*

The last character contains the hexadecimal character F to ensure a valid numerical sign. The remaining characters are:

c = century indicator (c=0=1900; c=1=2000; etc.)
yy = year indicator (00 to 99)
ddd = day indicator (001 to 366)

Return Codes

A return code of 0 is issued upon normal completion of the DATE function.

Abends

An abnormal termination may occur during execution of the DATE function. A possible cause is that an invalid *area* argument was specified.

17 EOJ Function

The EOJ function is used to initiate Com-plete end-of-job processing for an online or batch program.

> **Note:** If a batch program terminates without issuing an EOJ function, the session not terminated by the target Com-plete.

Either the EOJ or the WRTD function of Com-plete can be used to terminate the program normally. The WRTD function differs from the EOJ function in that it permits the application program to issue a termination message.

Format

The format for using the EOJ function is:

```
EOJ
```

No arguments are provided or Required.

Return Codes

No return codes are given by the EOJ function. Since end-of-job processing is invoked, the user program will not regain control.

Abends

In case of an abnormal termination, consult the system programmer.

18 FREEMAIN Function

The FREEMAIN function is used to free storage from the thread in which the application is running. Note that the storage to be freed must have been acquired by a previous GETMAIN or GETSTOR.

The length and area values must always be double word multiples. For a FREEMAIN request, Com-plete rounds the length value "UP" and sends a return code 12 to the user program if the area address is not aligned.

Note that the length and address values specified should be those returned by a previous GETMAIN or GETSTOR.

Format

The format for using the FREEMAIN function is:

```
FREEMAIN (retcode, length, area)
```

retcode	Required. Specifies a fullword where Com-plete will place the return code upon completion of the operation.
length	Required. Specifies a fullword containing the length of the area to be freed.
area	Required. Specifies a fullword containing the address of the area to be freed.

Return Codes

The contents of Register 15 should be checked for the following conditions:

0	No errors.
8	The request space was not previously acquired.
12	An invalid request or an invalid FQE was found.

Abends

There are no abends associated with the FREEMAIN function.

19 GETCHR Function

The GETCHR function enables an online or batch application program to obtain information about the terminal environment in which it is executing or information about the terminal environment for a specified terminal. The information obtained is returned in the form of a table of information referred to as the GETCHR information table. Note that the GETCHR information table should be defined in the working storage area of the application program.

Three arguments are provided by the GETCHR function. The information obtained is placed in a working storage area in the application program indicated by the required area argument. The information supplied is arranged into separate fields, each of which has a length of one to eight bytes.

The optional *length* argument is used to specify the number of bytes to be returned to the program. If the *length* argument is not used, Com-plete returns only the first sixteen bytes of information. If specified, the *length* argument value must end on a GETCHR field boundary.

The optional *tid* argument enables specification of the terminal to which the GETCHR information pertains. If the TID is omitted, the GETCHR function provides information only for the terminal in use.

The format and content of the GETCHR record are illustrated in **Getchr Information Table**.

Format
The format for using the GETCHR function is:

```
GETCHR (retcode,area[,length][,tid])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
area	Required. The GETCHR information table area where Com-plete is to place terminal environment information.
length	Optional. Default: If the length is omitted, 16 bytes of information will be provided.A binary halfword containing the number of bytes of terminal environment information to be returned. The contents of length must be a number that ends on a GETCHR field boundary. The length can contain a maximum value of 512.
tid	Optional. Default: If omitted, the environment information returned will be for the TID of the terminal in use. A binary halfword containing the Terminal Identification number (TID) or the Line Identification number (LID) of the terminal or line group about which environment information is being requested.

Return Codes

The following return codes are issued by the GETCHR function:

0	The information returned was for a TID.
4	The information returned was for a LID.
8	The TID/LID number specified exceeds the maximum defined in the TIBTAB.
12	The TID/LID number specified does not exist.

Abends

An abnormal termination may occur during execution of the GETCHR function. Possible causes include:

- An invalid argument was specified on the GETCHR call;

- A protection exception occurred.

20   GETMAIN Function

The GETMAIN function is used to acquire storage from the thread in which the application is running. This storage can be used for any function desired by the program.

The amount of storage requested is acquired only if that amount of storage is available within the thread as a contiguous segment. Note that there is no protection from storage overruns within the thread.

When a request is made, the largest contiguous segment available that is greater than or equal to the minimum and less than or equal to the maximum will be returned.

Note that requested lengths must be in double word segments, or Com-plete will round up length requests to double word multiples. This action on the part of Com-plete is consistent with that of the FREEMAIN function as well, and prevents storage fragmentation to some degree.

Format

The format for using the GETMAIN function is:

```
GETMAIN (retcode,minmax,addrlen)
```

retcode	Required. Specifies a fullword where Com-plete places the return code upon completion of the operation.
minmax	Required. Specifies the address of two binary fullwords. The first of these words contains the minimum amount of storage that will satisfy the request. The second specifies the maximum amount of storage needed by the request.
addrlen	Required. Specifies the address of two binary fullwords. The address of the acquired storage is placed into the first word. The length of the acquired storage is placed into the second word.

Return Codes

The following return codes are issued by the GETMAIN function:

0	No errors.
4	The requested space was not available.
12	An invalid request or an invalid FQE was found.

Abends

There are no abends associated with the GETMAIN function.

21 **GETSTOR Function**

The GETSTOR function is used to acquire storage from the thread used by the application. In contrast to the GETMAIN function, GETSTOR can be used to allow storage to be obtained from above the 16 MB line, if available.

The length requested must be a multiple of eight, otherwise Com-plete will round it up.

Storage acquired using the GETSTOR function can be released using the FREEMAIN function.

Format

The format for using the GETSTOR function is:

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
length	Required. A fullword containing the length of the storage area to be acquired.
address	Required. A fullword where Com-plete places the address of the acquired storage.
location	Optional. Default: If no location is specified, storage is acquired from below the 16 MB line. A three-byte character field indicating where to acquire storage from: BEL indicates storage to be acquired from below the 16 MB line, ANY indicates storage to be acquired from any location above or below the 16 MB line.

Return codes

0	Successful completion.
4	The amount of space requested was not available.
12	An invalid request or an invalid FQE was found.

22 LEPIPI Function

The LEPIPI function can be used to call LE routines from non-LE programs. When it receives the first LEPIPI call from a given application, Com-plete uses LE preinitialization services (CEEPIPI init_sub_dp) to initialize a HLL environment. This environment is then maintained for subsequent calls, until a termination call is received, or the application terminates.

The routines being called must match the requirements defined by the IBM Language Environment. Both main routines and subroutines are supported.

Runtime options for IBM Language Environment modules invoked using this function can be specified with the LEPIPI-RUNTIME-OPTS sysparm.

Format

The format for the LEPIPI function is:

```
LEPIPI (retcode, module, plist, return_vector)
```

retcode	A fullword where Com-plete places the return code from CEEPIPI call_sub_addr (or term). The value is also returned in R15.
module	A character string of length 8, left-justified and padded right with blanks, containing the name of the LE routine to be called. The routine must match the requirements defined by the IBM Language Environment. Both main routines and subroutines are supported. A NULL pointer in this parameter is treated as a request to terminate the PIPI environment.
plist	The (direct) address of a parameter list to be passed to the LE routine.
return_vector	The address of 5 consecutive fullwords where LE places:
	■ the 4 byte enclave or subroutine return code,
	■ the 4 byte enclave or subroutine reason code,
	■ the 12 byte enclave feedback code.

Return codes

Com-plete passes back the return code from CEEPIPI call_sub_addr (or term). In addition to those, a return code of 20 (dec) is used to indicate a "module not found" condition.

23 **MODIFY Function**

The MODIFY function allows an application program to change several of the processing characteristics associated with it and with the terminal. For example, the MODIFY function could be used by an application to change the definition of its terminal from upper-case to lower-case.

Format

The format for using the MODIFY function is:

```
MODIFY (retcode,function[,option])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
function	Required. A four-character field containing the code indicating the function to be performed. Valid function codes are described in the function code table below.Note that codes less than four characters are left-justified and padded with blanks to the right.
option	Optional. An additional parameter that can be used by certain functions. The format of this parameter is described in the function code table below.

Return Codes

The following return codes are issued by the MODIFY function:

0	The function completed normally.
4	The request could not be completed.
8	The request is invalid.

Abends

There are no abends associated with the MODIFY function.

Function Codes

The following table defines the valid MODIFY function codes and the actions associated with each. Also described are the optional parameters, if any, associated with each function code.

Code	Function	Optional Parameter
AFF+	Increase the TCB affinity level of the current process by 1. The TCB affinity level is the number of reasons why a process must not switch TCBs. If it is 0 (which is usually the case), then the process can continue executing on a different TCB after an interrupt. Com-plete automatically maintains the affinity level for well-known events such like ENQ - DEQ. **Note:** This function code should only be used by experienced users.	None.
AFF-	Decrease the TCB affinity level of the current process by 1. **Note:** This function should only be used if your program has successfully invoked AFF+ before.	None.
UP	Set the terminal into upper case mode	None.
LOW	Set the terminal into lower case mode	None.
HC	Change the screen-to-hardcopy definition	An eight-character name of the new screen-to-hardcopy device. If it is omitted, screen-to-hardcopy is reset.
REL	Set the program to be relocatable.	A halfword mask defining which registers are to be relocated with the program.
NREL	Set the program to be non-relocatable	None.
ALT	Set the terminal to use alternate screen size.	None.
NALT	Set the terminal to use normal screen size.	None.
TCS	Set the program protection key to Com-plete's protection key. This function can be used only from PV and/or resident programs.	None.
THRD	Set the program protection key to the normal user protection key. This function can be used only from PV and/or resident programs.	None.
TRID	Change the transaction ID (name) of the current transaction.	An eight-character field containing the new transaction ID.
PRTY	Set the transaction priority.	A fullword containing a new priority value between 0 and 3.
EXTD	Set the terminal to use extended data stream support.	None.
NXTD	Set the terminal to not use extended data stream suppor	None.

Code	Function	Optional Parameter
COMP	Set data stream compression for the terminal on.	None.
NCMP	Set data stream compression for the terminal off.	None.
NAME	Set the stack name for the current level to the supplied value.	An eight character field.
KANO	Set KANJI mode off.	None.
KANI	Set KANJI mode to type IBM.	None.
KANH	Set KANJI mode to type HITACHI.	None.

24   RJE Function

The RJE function enables an online or batch application program to submit jobs to the operating system for scheduling and execution in the batch environment. Card input via a batch job input stream is simulated through use of an internal pseudo card reader defined to Com-plete.

Job stream data to be submitted is stored in the working storage area of the application program. When the job stream data is submitted with the RJE function, the amount of data must be specified in bytes. The number specified must be a multiple of 80.

The application program using the RJE function may have a need to submit more job stream data than can contiguously be stored in the working storage area of the application program at any one time. Consequently, submission of the job stream data might require more than one use of the RJE function to, in effect, submit portions of the job stream data. This feature is provided for by the RJE function.

Format

The format for using the RJE function is:

```
RJE (retcode,area,length[,options])
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
area	Required. A buffer area in the working storage area of the application program that contains the job stream data to be submitted.
length	Required. A binary halfword containing the length of the job stream data, in bytes, to be submitted. This value must be a multiple of 80.

options	Optional.
	A binary halfword containing the processing options to be invoked at the time of RJE submission.
	"X8000" indicates that the internal reader will be held for additional RJE submission by the currently executing program after execution of the RJE function. The absence of this argument indicates that the reader is to be made available immediately for use by other programs.
	If the application terminates, Com-plete will free the reader (if held). However, the reader is not freed over a screen I/O or rollout function. This may cause operational problems, so please be cautious about using the HOLD option.

Return Codes

The following return codes are issued by the RJE function:

0	The job stream data was successfully submitted.
4	The job stream was not successfully submitted. In this situation, the pseudo card reader defined to Com-plete is in use by another application program. The application program receiving a return code value of 4 should normally reissue the RJE request. Continued receipt of a return code 4 can indicate a potential operational problem, notify the system programmer.
8	The RJE functions have been disabled.
12	RJE submit aborted by user exit.

Abends

An abnormal termination may occur during execution of the RJE function. Possible causes include:

- An invalid *area* or *length* argument was specified;

- The *length* argument is not a multiple of 80.

25 **ROLEVT Function**

The Roll-for-Event (ROLEVT) function provides a facility that allows the application program to rollout of the thread until an event controlled by a companion JOB (or task) occurs. This facility allows a Com-plete application program to interact with another JOB running in the same machine and provides a synchronization mechanism between the Com-plete program and the companion JOB.

Synchronization of the two tasks is controlled through an Event Control Block (ECB), which is waited on by the ROLEVT function, and which must be posted by the companion task to signal a synchronization. ROLEVT uses a standard operating system WAIT macro to await the event occurrence, and relies on the companion task to issue the operating system POST macro to signal the event occurrence.

The ROLEVT function can be used by any online Com-plete application program, privileged or non-privileged. However, a distinction is made by the ROLEVT function between privileged and non-privileged programs when validating the ECB argument.

For non-privileged programs, the area pointed to by the ECB argument must be located within storage acquired via the Com-plete COMSTOR function. This area is located within the Com-plete partition/address space, but outside of the Com-plete threads. For privileged programs, no address validation is performed.

If the ROLEVT request is rejected, the program is not rolled out and remains in the thread. However, once rolled out as a result of the ROLEVT function, the application remains rolled out until the companion task POST's caller's ECB occurs. The only exception to this is if the application program is cancelled by the computer operator with the operator command CANCEL. When this happens, the application program is rolled back into the thread and terminated abnormally.

Format

The format for using the ROLEVT function is:

```
ROLEVT (retcode,CB)
```

The argument is:

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
CB	Specifies the address of a control block to be used to pass information back and forth between the application program and Com-plete. The first word of the control block contains the address of an ECB. The ECB pointed to is used by Com-plete to wait for the event occurrence to be signalled by a POST from a companion task. This area is also used by Com-plete to pass back the status of the ROLEVT function.The area pointed to by this argument is validated by the ROLEVT function, according to the rules established earlier in this section.

Return Codes

The return code parameter or the first word of the control block must be checked for the following conditional values:

0	ROLEVT request accepted. The ROLOUT and subsequent ROLIN has been completed.
4	Invalid control block address specified.
8	COMSTOR does not exist (non-privileged programs).
12	ECB address is not within an area acquired with the Com-plete COMSTOR function (non-privileged program).
16	The specified ECB has already been waited on.

Abends

The only abnormal termination associated with the ROLEVT functions is one indicating that the parameter list is invalid.

26 **ROLOUT Function**

The ROLOUT function enables an application program to relinquish the use of the thread by requesting that a thread rollout occur. Thread rollouts are required to enable the sharing of thread resources by more than one application program.

When Com-plete is initialized, a maximum CPU time value is assigned to each thread. If an application program uses this maximum value before a rollout occurs, the program terminated abnormally.

Thread rollout normally occurs when an application program uses a terminal I/O function (e.g., WRTC); however, if an application program performs multiple compute-bound or I/O-bound functions prior to using a terminal I/O function, it may exceed the time allotted and abnormally terminate. Use of the ROLOUT function enables the application program to avoid abnormal termination by forcing the thread rollout to occur.

When a thread rollout occurs, the program in the thread is placed at the end of the Com-plete ready-to-run queue, and the application program at the top of the queue is dispatched for execution. If the ready-to-run queue is empty (no program is awaiting execution), a rollout does not occur. Note that this is true even if the ROLOUT function is specifically requested.

The application program can optionally choose to request that the thread rollout occur and that (after a specified number of seconds) the application program be placed at the bottom of the Complete ready-to-run queue to await dispatching for execution. This is called a timed rollout request.

Format

The format for using the ROLOUT function is:

```
ROLOUT [(time)]
```

time	Optional. Default: 0A binary halfword containing the number of seconds after which the application program is placed at the bottom of the Com-plete ready-to-run queue to await dispatching for execution.

Return Codes

There are no return codes associated with the ROLOUT function.

Abends

There are no abnormal termination situations that can be directly associated with the ROLOUT function.

27 SETEID Function

The SETEID function is used to enable an application program to recognize the entry of a Program Attention (PA) key or a Program Function (PF) key by the terminal user.

Some PA and/or PF keys can be reserved for Com-plete functions via system parameters or by user profile definition.

If the PA/PF key entered is allocated to one of the Com-plete functions SUSPEND, HARDCOPY or JUMP then this key is not passed to the application and the appropriate Com-plete function is performed.

If the **PA2** key is allocated to the Com-plete HARDCOPY function, then this key is passed to the application if the system parameter definition was OVERRIDE and the application requested that the **PA2** key be passed.

The application program can assign a function of its own to one or more PA or PF keys. In order to facilitate recognition of one of these keys, the keys must be masked off from Com-plete. This is accomplished by using the SETEID function. Once the function keys are masked off from Com-plete, the application program must issue a terminal device-dependent read and examine the first character in the input buffer to determine which function key, if any, was pressed.

An application program can issue more than one SETEID function.

Format

The format for using the SETEID function is:

```
SETEID (retcode,eid)
```

retcode	Required. A fullword where Com-plete places the return code upon completion of the operation.
eid	Required. A binary halfword field. Each bit position within the field represents a PAn key or a PFn key.
	If a bit position is 1, the associated PA/PF key entry will be returned to the application program. Note that the adding together of values associated with a key produces an eid value necessary for masking those keys.
	Use the following table to determine which bit affects the appropriate key:

Key	Bit	Eid
PA1	0	32768
PA2	1	16384
PA3	2	8192
PF1	3	4096
PF2	4	2048
PF3	5	1024
PF4	6	512
PF5	7	256
PF6	8	128
PF7	9	64
PF8	10	32
PF9	11	16
PF10	12	8
PF11	13	4
PF12	14	2
ALL	-	specifies a mask of X'FFFF'
NONE	-	specifies a mask of X'0000'

Note: Specifying PF1 also specifies PF13, PF2 specifies PF14, etc.

Return Codes

A return code of 0 is issued upon normal completion of the SETEID function.

Abends

An abnormal termination may occur during execution of the SETEID function. A possible cause is that an invalid *eid* argument was specified.

28 SNAP Function

The SNAP function can be used to write a thread dump of the application, without terminating the application. The purpose of this function is to help debugging applications.

Format

The format for the SNAP function is:

```
SNAP (retcode,header,retnumber)
```

retcode	A fullword where Com-plete places the return code.
header	address of a 77 byte character string to become the dump header. First 7 bytes are recommended to contain a message ID to be shown in UDUMP ALL (UDUMP will prefix it with "USR", indicating that this dump was produced using the SNAP function).
retnumber	A fullword where the dump number is returned upon successful completion.
arealist	Optional. A fullword containing the address of a list of up to six 16-byte entries describing areas of memory to be included in the dump. Each entry must be built according to the following structure: 1. a fullword containing the length of the area; 2. a fullword containing the starting address of the area; 3. an eight byte character string identifying the area. The table must be terminated by an entry with a zero length field.

Return Codes:

0	dump taken
8	no dump taken (suppressed or error)

Abends:

There are no abends expected with the normal usage of this function.

29 TESTAT Function

The TESTAT function is used to determine whether or not the terminal operator has caused what Com-plete considers to be an attention interrupt condition.

Since the TESTAT function tests for an attention interrupt condition, it follows that only terminal device types that generate an attention interrupt can be used with programs that use the TESTAT function. Normally the device types are restricted to 3270-type terminals (ENTER key) and TTY-terminals (BREAK key).

Format

The format for using the TESTAT function is:

```
TESTAT (retcode)
```

Return Codes

The following return codes are issued by the TESTAT function to indicate the attention interrupt status caused by the terminal user:

0	An attention interrupt was not received.
4	An attention interrupt was received.
8	The terminal device in use is not an attention interrupt device.

Abends

There are no abnormal termination situations that can be directly associated with the TESTAT function.

30 TIME Function

The TIME function enables the application program to obtain information about timing intervals in one of three precision levels:

- Timer units
- Binary
- Decimal

If time is requested in timer units, an unsigned fullword containing the number of timer units elapsed since midnight is returned to the application program. A timer unit is defined as 26.04166 microseconds.

If time is requested in binary, a positive fullword containing the number of hundredths of seconds since midnight is returned to the application program.

If time is requested in decimal, a signed fullword of the format X'hhmmssth' is returned to the application program (hours, minutes, seconds, and tenths of seconds since midnight).

The time can also be obtained by using the standard COBOL reserved word TIME-OF-DAY or the standard PL/I functions.

Format
The format for using the TIME function is:

```
TIME (retcode,area,code)
```

retcode	Required.
	A fullword where Com-plete places the return code upon completion of the operation.
area	Required.
	A fullword area in the working storage area of the application program where the time returned.
code	Required.
	A binary halfword containing the code that specifies the type of time being requested. Permissible values for the code are 0, 1, or 2.
0	indicates timer units.
1	indicates binary units.
2	indicates decimal units.

Return Codes

A return code of 0 is issued upon normal completion of the TIME function.

Abends

An abnormal termination may occur during execution of the TIME function. Possible causes include:

- An invalid *area* argument was specified;

- An invalid *code* argument was specified;

- The *area* argument field is not on a fullword boundary.

VI Function Tables

This part of the Application Programming documentation provides presents various control blocks and code tables as well as additional technical information concerning the values you have to use to customize API functions.

This information is organized under the following headings:

Mapping Request Control Block (MRCB)

Mrcb Exception Codes

Field Control Table (FCT)

Field Descriptor Codes

Terminal Control Codes

Request Parameter List

Captur Record Header

Message Switching Control Block (MESGCB)

Printout Spool Control Block (PSCB)

GETCHR Information Table

Com-plete Functions For Batch And Online Programs

Terminal Device Type Codes

31 Mapping Request Control Block (MRCB)

The MRCB is a working storage area defined within the application program area used for processing mapping requests. It contains the name of a map to be used in a Com-plete mapping CALL function, plus additional control information pertinent to both input and output operations.

The format of the MRCB is shown in the following table.

Copy code for the MRCB is provided in the Com-plete source library:

- CCMRCB - Assembler
- COBMRCB - COBOL
- PL1MRCB - PL/I

Field Name	Offset	Len	Description
The first 8 bytes should be:			
MAPNAME	0	4	Required.
VERSION	4	1	Required. If B, then MAPMVER must be blank.
FILLER	5	3	Must be spaces.
or:			
MAPNAME	0	6	Required. Six-character map name must have device code in the last two bytes (for example, F2).
FILLER	6	2	Must be spaces and MAPMVER must be B.
MAP-COUNT	8	2	Binary halfword. Specifies the number of maps to be kept resident. The default value, spaces, is the same as 2 (binary 16448, x'4040'). The map count field should be reset to zero (x'0000') on the last mapping call that uses a map so that the map is deleted from thread storage. Failure to do this may result in an abend S80A of the application, due to insufficient storage.

Field Name	Offset	Len	Description
FCTE-COUNT	10	2	Binary halfword. Specifies the number of Field Control Table Entries (FCTEs) in the FCT. Must be less than 16449. A value of 16448 (x'4040') is the same as 0.
FCE-FORMAT	12	1	Character. Must be one of the following: S or space FCT entries are short form entries of 6 bytes each. They contain the field name only. L FCT entries are long form entries of 10 bytes each. They contain the field name, an input flag, and the FDC override field. E FCT entries are extended form entries of 13 bytes each. They contain the field name, an input flag, the FDC override field, the color override field, and the symbol set ID override.
MAP-CONCAT	13	1	Character. Must be one of the following: N or space The map is not concatenated or addressed. The map is accessed from the resident area of Com-plete or the Com-plete load library chain. A The map is addressed, and the location is specified by MAP-ADDRESS. C The map is concatenated to the MRCB and does not need to be loaded.
WRITE-OPTION	14	1	Character. Must be one of the following: A or space Write all unprotected fields. O Write only those fields specified in the FCT.
READ-OPTION	15	1	Character. Must be one of the following: A or space Allow all unprotected fields that have been entered by the terminal operator to be moved into the data area. O Allow only those unprotected fields specified in the FCT and those that have been entered by the terminal operator to be moved into the data area.
TCC	16	8	Terminal control codes to be concatenated to those in the map.
CURSOR-OUT	24	6	Field name to place cursor in during WRTM call.
CURSOR-IN	30	6	Name of the field that the cursor was found in during the READM call.
RETURN-CODE	36	2	Binary halfword. Return code from the last READM or WRTM call.
ENTER-CODE	38	2	Alphanumeric codes or numeric codes to indicate which terminal interrupt key was pressed. With the exception of the alphanumeric codes TR and SP, the alphanumeric codes signify those keys that do not transmit data. Numeric codes indicate data transfer, if data was entered. space No data entered. CL Clear key entered. TR Test request entered. SP Selector pen interrupt. SC ID card reader. A1 through A3 through keys entered. 00 Normal entry (ENTER, EOT, etc.). 01 through 24 through keys.
FIELDS-READ	40	2	Binary halfword. Count of the number of fields read.

Field Name	Offset	Len	Description
ERROR-FIELDS	42	2	Binary halfword. Count of the number of input fields found in error (invalid keyword, alpha data in numeric field, etc.). This field can be multiplied by 10 to obtain the length for writing the feedback area to the terminal.
FEEDBACKLGTH	44	2	Binary halfword. Specifies the length of the MRCB feedback area. This value must be less than 16449. 16448 (x'4040') is the same as 0.
MAPMVERS	46	1	Space or scaling of map. Mapping uses the map with name built from four characters of the MAPNAME field concatenated with the device code of the terminal.
			B Scaling of map. Mapping uses the map with the name given by all six characters of the MAPNAME field. The map is scaled to fit the device code of the terminal in use.
	47	1	Must be spaces. Reserved for future use.
MAP-ADDRESS	48	4	Must be spaces unless MAP-CONCAT is A. If MAP-CONCAT is A, then this is the address of the map. The map can be located within the thread, or in the resident area of Com-plete.
EXPANSION	52	18	Must be spaces. Reserved for future use.
FEEDBACK-AREA	70	nn	Feedback area used during input exception processing. If this area exists, the MRCB field FEEDBACK-LGTH must be non-zero and less than 16448. When an input exception occurs, the name of the field in error and an exception code are placed here. The format of each entry is "FFFFFF XX", where "FFFFFF" is the name of the field in error, and "XX" is the exception code.

32 Mrcb Exception Codes

The Mapping Request Control Block (MRCB) exception codes are used to indicate input errors from a terminal. Data entered at a terminal that conflicts with the field definition for the mapping field in which it is entered is not returned in the buffer area. Instead, the name of the mapping field, followed by an exception code, is listed in the feedback area.

Error Code	Field Types
ND	All
NN	Z
NN	P,H,F
OF	All
UF	P,H,F
MR	All

33 Field Control Table (FCT)

The Field Control Table (FCT) is a storage area defined within the application program area and used during map processing. It is used by the application program to alter the display characteristics of individual fields during output processing and to receive additional information about each field during input processing.

The entries in a given FCT are called Field Control Table Entries (FCTEs). There are three forms of FCTEs, only one of which will exist in any one FCT:

- Short form
- Long form
- Extended form

The short form of the FCTE has only:

```
FIELD-NAME
```

The fields in the long form FCTE are:

```
FIELD-NAME
INPUT-FLAG
FDC-OVERRIDE
```

The fields in the extended FCTE are:

```
FIELD-NAME
INPUT-FLAG
FDC-OVERRIDE
COLOR-OVERRIDE
SYMBOL-SET-OVERRIDE
```

FCTE fields are described in the following table.

Field Name	Offset	Len	Description
FIELD-NAME	0	6	The name specified must be the same as that specified for a field name within the map being used. If no match is found, the name is ignored.
INPUT-FLAG	6	1	Space Input was entered and is valid. N No data entered for this field. I Invalid data entered. O Overflow. Too much data entered. U Underflow. Too many decimal places entered S Null entry. The field was entered with no data. This can be caused by use of the selector pen, or by entering a keyword with no data. In either case, an FDC of L must have been specified for this field when the map was built, or in the FDC-OVERRIDE entry of the FCTE. D The DUP key was pressed for this field.
FDC-OVERRIDE	7	3	The FDCs entered here are concatenated to the FDCs specified for the field in the map. The FDCs is then processed. The last FDC specified takes precedence, if conflicting FDCs exist. A special FDC of I can be placed in any position of this field and causes this FCTE to be ignored. FDCs are fully described in **Field Descriptor Codes**
COLOR-OVERRIDE	10	2	The two-character color override replaces the map's color attribute for this field.

BL	blue
RE	red
PI	pink
GR	green
TU	turquoise
YE	yellow
NE	neutral
or blank	neutral

Field Name	Offset	Len	Description
SYMBOL-SET-OVERRIDE	12	1	The symbol set override replaces the map's SYMBOL-SET-ID for this field.

34 Field Descriptor Codes

The Field Descriptor Codes (FDCs) are used during map processing. They are specified in the map during map creation or they can be used during program execution.

When used during map creation, FDCs are entered in either the MAPSTART definition, the MAPF definition, or with the ATTRIBUTE UPDATE function of the UMAP utility. During program execution, they are specified in individual FCTE entries.

The permissible FDCs are defined in the following table. The FDCs are grouped in six categories. The FDCs in each category are mutually exclusive. Note that if more than one code from any one group is specified, the last one encountered takes precedence.

FDC	Field Type	Definition
(1)		
B	All	Blink. (If the device type supports the feature.)
D	All	Display with normal intensity
H	All	High intensity display
L	Variable only	Light pen selectable
N	All	Non-displa
V	All	Video. Reverse the display in the field if the video feature is supported by the device.
X	All	Underline the field if the feature is supported by the device.
(2)		
P	All	Protected. Do not allow data to be entered in this field.
S	All	Skip. Do not hardware tab stop on this field.
U,T	All	Unprotected tab. Do allow this field to be modified by setting the hardware tab.
(3)		
O	All	Optional. Data entry is optional in this field.
R	Variable only	Required. Data entry is required in this field.

FDC	Field Type	Definition
(4)		
K	All	Turn off the modified data tag in this field.
M	Variable only	Turn on the modified data tag in this field. **Note:** M is supplied as a convenience to 3270 users only. It's use by application devices other than 3270 may yield unpredictable results. When conversation mode is used, terminal I/O should eliminate the need for using the modified data tag to store data at the terminal.
(5)		
Y	All	Skip can be used. The field can be used with a skip character. The skip character will not overlay a field already on the screen. Used for split screen mapping. (Constant fields require no ending attribute.)
Z	Variable only	Skip can not be used. The field can not be ended with a skip character because the attribute byte will overlay a field already on the screen. Used for slip screen mapping.
(6)		
E	Variable only	Erase the field.

35 Terminal Control Codes

The Terminal Control Codes (TCCs) are used to specify terminal control options during use of Com-plete mapping functions. They can be specified during map creation and/or during program execution.

During map creation, TCCs are defined in the MAPSTART macro statement. During program execution, they are specified dynamically in the TCC field of the MRCB.

Valid TCCs are defined in the following table.

TCC	Definition
A	Sound the audible alarm.
B	Allow Com-plete to determine whether the screen should be erased before the write.
C	Always format the screen with constant fields for this map.
D	Do not format the screen. Even though a new map is requested, the format will not be written.
E	Erase unprotected fields.
F	Allow Com-plete to determine whether the screen is to be formatted.
K	Turn off all modified data tags.
L	Do not reset the keyboard.
M	Do not turn off modified data tags.
N	Do not erase unprotected fields.
P	Start the printer.
Q	Do not sound the audible alarm.
R	Reset the keyboard.
S	Do not start the printer.
W	Do not erase the screen before writing the format (constant fields).

> **Note:** The use of the TCC M is supplied as a convenience to 3270 users only. Its use by applications with devices other than the 3270s may yield unpredictable results. The use of the Com-plete conversational mode terminal I/O feature eliminates the need for using modified data tags to store data at the terminal.

For formattable devices, the default TCCs are E, R, Q, S, B, F, and K.

The TCCs B and W function independently of codes E and N. Codes E and N are used to erase unprotected fields, while codes B and W are used to control the erasure of the entire screen format.

36   Request Parameter List

The Request Parameter List (RPL) is a working storage area defined in an application program and is used with Com-plete file I/O functions when accessing BDAM or ISAM data sets.

A separate RPL can be defined for each file to be accessed, but only one RPL is required and can be shared by separate files, if the application program performs the necessary initialization functions. However, if more than one file is to be accessed simultaneously, separate RPL definitions must be established.

The format of the RPL is shown in the following table.

Copy code for the RPL is provided in the Com-plete source library: COBRPL45 for COBOL, and PL1RPL45 for PL/I. For reasons of compatibility, the RPL format used in previous releases of Com-plete remains available.

Fieldname	Location Dec	Location Hex	Length	Format	Contents
RPLIDENT	0	0		Character	Identifier string '0450'.
RPLDDN	4	4	8	Character	DDNAME for file.
RPLNOREC	12	C	2	Binary	Number of records to be processed. O is same as 1.
RPLINERR	14	E	2	Binary	Record in error. Used only when processing more than one record. Contains number of record in error when I/O error occurs, relative to number requested.
RPLOCNT	16	10	2	Binary	Number options. Indicates how many of the next two fields are to be included in the call. 1-Include next field 2-Include next two fields
RPLACC	18	12	3	Character	Type access. Specifies the technique to be used for accessing a record: SEQ Sequential / DIR Direct / BGN First record
RPLOPTN	21	15	3	Character	Key option. Used for ISAM files only: KEQ Key equal / KGE Key greater or equal

Type Access Field

The type access field, located at relative location 18 (X'12'), indicates the processing technique to be used when accessing records from the file. The options are listed in the following table.

Option	Explanation
SEQ	Specifies that records are to be accessed sequentially. The key option field should not be specified. For ISAM file I/O requests, the key option field is ignored. For BDAM file I/O requests, the application program is abended if the key option field is specified.
DIR	Specifies that records are to be accessed randomly. For ISAM file I/O requests, the number options field should be 2, and the key option field should be initialized. For BDAM file I/O requests, the number options field should be 1. If the key option is specified, the application program is abended,. This is the only allowable option if accessing BDAM files using absolute addressing (MBBCCHHR).
BGN	Specifies that the first record in the file is to be accessed. If more than one record is to be processed with this call, the number of records requested is processed beginning with the first record in the file. Note that the key option field, located at relative location 21 (X'15'), is used for ISAM file accesses only. If specified for a BDAM file, the application program is abended.

Key Option Field

The key option field is specified only when records are to be processed randomly by key. In this case, the number options field must be initialized to 2. If the key option field is not to be used, the number options field must always be initialized to 1.

The options are listed in the following table.

Option	Explanation
KEQ	Specifies that the record in the file with a key equal to that of the key supplied is to be accessed. If the key option is not specified, this is the default.
KGE	Specifies that the record in the file with a key equal to or greater than that of the key supplied is to be accessed.

37 CAPTUR Record Header

Capture records are written to the Com-plete capture data set by use of the CAPTUR function. These records can subsequently be processed using the CUCTCAPT batch utility program.

Each record written to the capture data set is prefixed by a variable-length header generated by Com-plete. The format of this header is indicated in the following table.

Location		Length	Format	Contents	
Dec	Hex				
0	0	1	Binary	Record identifier:	
				X'01'	Online user program.
				X'02'	Batch user program.
				X'03'	Reserved.
				X'04'	TCSLOG record.
				X'06'	Reserved.
				X'07'	DASD security password.
				X'09'	File I/O record.
				X'D3'	Capture label record. See the Com-plete System Programming documentation for the format.
1	1	3	Decimal	Date in the format yydddF.	
4	4	3	Binary	Time in 100ths of seconds.	
7	7	4	Binary	X'FFFFFFFF'	
11	B	3	Binary	Version|Release|SMlevel	
14	E	8	Character	Program name or user supplied ID	
22	16	2	Binary	TID number	
24	18	4	Packed	Date in the format 0cyydddF	
28	1C	4	Binary	Reserved for Software AG.	

38   Message Switching Control Block (MESGCB)

The Message Switching Control Block (MESGCB) is a working storage area in the application program containing the information Com-plete needs to control the processing of the MESGSW switching function. The format and content of the MESGCB follow:

Location		Length	Format	Contents
Dec	**Hex**			
0	0	1	Character	L (=Last segment) space
	1	1	Character	Reserved. Must be initialized to a space.
	2	2	Binary	Message identification number. Must be initialized to spaces. Com-plete inserts the message number assigned to the message when the first segment is sent.
	4	2	Binary	Class codes. See the following table.

The class codes specified in the MESGCB are defined as bit settings in the halfword located at relative offset four. The following table details the bit settings associated with specific class codes:

BIT POSITIONS

Class Code	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.
4	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.
5	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
6	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.

8	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.
9	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.
10	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.
11	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.
12	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.
13	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.
14	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.
15	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.
16	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Copy code for the MESGCB is provided in the Com-plete source library: COBMSCB for COBOL, and PL1MSCB for PL/I.

39 Printout Spool Control Block (PSCB)

The Printout Spool Control Block (PSCB) is a working storage area in the application program containing the information Com-plete needs to control the processing of printout spool function requests. The format and content of the PSCB follow:

Location		Length	Format	Contents
Dec	Hex			
0	0	2	None	Reserved. Must be initialized to spaces.
2	2	2	Binary	Reserved. Used by Com-plete to contain the printout spool identification number. Must be initialized to zeros.
4	4	2	Binary	Printout spool class codes indicated as a binary halfword bit map.
6	6	2	Binary	Logical output statement length (record length).
8	8	4	Character	Constant: EXT1
12	C	8	Character	Listname before Open.
12	C	4	Binary	Address of MCQ after Open.
20	14	4	Character	Form-ID.
24	18	1	Character	Disposition of printout:
				D Print and delete after print.
				H Hold printout until release.
				L Print and hold after print.
25	19	1	Binary	Reserved
26	1A	8	Character	Output logical driver name.
34	22	1	Binary	Number of additional copies.
35	23	5	None	Reserved.

The class codes specified in the PSCB are defined as bit settings in the halfword located at relative offset four. The following table details the bit settings associated with specific class codes:

BIT POSITIONS

Class Code	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.
4	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.
5	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
6	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.
8	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.
9	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.
10	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.
11	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.
12	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.
13	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.
14	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.
15	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.
16	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Copy code for the PSCB is provided in the Com-plete source library: COBPSCB for COBOL, and PL1PSCB for PL/I.

40 GETCHR Information Table

The GETCHR information table is defined in an application program as a working storage area and is initialized by executing the GETCHR function.

The information returned from the GETCHR function is terminal environment information.

Label	Length	Format	Contents
TMGETTID	2	Binary	TID.
TMGETLLN	2	Binary	Maximum line length.
TMGETMDL	2	Binary	Maximum device buffer length.
TMGETMLN	2	Binary	Maximum number of lines.
		0	hard copy device
TMGETDEV	8	Character	Device type.
			See device type table in *Terminal Device Type Codes*
TMGETNAM	8	Character	Last called by terminal or fetch.
TMGETLDT	8	Character	Last load or fetched program.*
	2	Binary	Undefined.
TMGETOB	1	Character	Online/batch switch:*
			O	Online
			B	Batch
TMGETCTL	1	Character	TID status:
			C	Control
			U	Non-control TID
	4	Binary	Undefined.
TMGETCAT	4	Binary	Program catalog size.*
TMGETTRU	4	Binary	Program size.*
TMGETREG	4	Binary	Region size.*

Label	Length	Format	Contents
TMGETLPT	4	Binary	Program load point address.*
TMGETIID	8	Character	Installation ID.
	2	Binary	Undefined.
TMGETSCH	2	Binary	Default SCHC TID.
TMGETDTE	4	Packed	Julian date of ULOG ON.
TMGETNOD	2	Character	Access Node.
TMGETTNM	8	Character	TIBNAME.
TMGETUID	8	Character	User ID.
TMGETACC	12	Character	User ID account number.
TMGETAUT	2	Binary	User ID authorization code.
	3	Binary	Undefined.
TMGETLVL	1	Character	Current COM-PASS level number 1-9; 0, if not a COMPASS user.*
TMGETTIM	4	Binary	Time of ULOG ON in 100ths of seconds.
TMGETDTA	4	Binary	Amount of data sent and received by terminal since ULOG ON.
TMGETMSG	4	Binary	Amount of data sent via message switching and printout spooling since ULOG ON.
TMGETUSR	4	Binary	User field set by ULOGX1 and ULSRPSFS.
TMGETEXC	4	Binary	Number of EXCPs issued since ULOG ON on non-z/OS systems. For z/OS, number of SIOs issued since ULOG ON.
TMGETTRN	4	Binary	Number of transactions since ULOG ON.
TMGETTIL	4	Binary	Start time of current transactions, in 100ths of seconds.
TMGETCPU	4	Binary	Total CPU time elapsed since ULOG ON, in 100ths of seconds.
TMGETTHT	4	Binary	Total thread occupancy time since ULOG ON, in 100ths of seconds.
TMGETTID	2	Binary	Attaching TID.
TMGETNMP	2	Binary	Number of messages or printouts queued to terminal.
TMGETENT	4	Binary	Number of enters.
TMGETADC	4	Binary	Total Adabas calls.
TMGETADE	4	Binary	Total Adabas elapsed time.
TMGETADD	4	Binary	Total Adabas duration.
TMGETHNC	8	Character	Hardcopy device name.
TMGETIPA	4	Binary	TCPIP address.
TMGETIPP	4	Binary	TCPIP port.

* These fields apply only to the terminal executing the GETCHR function.

Copy code for the GETCHR area is provided in the Com-plete source; TMGETCHR for Assembler, PL1GCTBL for PL/1; COBGCTBL for COBOL.

41 Com-plete Functions For Batch And Online Programs

Com-plete allows the use of the following functions by batch programs, as well as online programs.

Function	Description
ABEND	Initiate Com-plete abnormal termination processing.
CAPTUR	Write user-specified data to the Com-plete capture log medium.
EOJ	Terminate a program, this function terminates Com-plete processing, but does not end the batch job step.
GETCHR	Obtain information about the terminal environment.
MESGSW	Send a message or a message segment.
PSCLOS	Logically close a printout spool data set.
PSOPEN	Provide to Com-plete information required to create a printout spool data set.
PSPUT	Output a record from a working storage buffer area to the printout spool data set.
RJE	Submit job streams to the batch environment for scheduling of execution.
SDCLOS	Logically close an SD file.
SDDEL	Delete specific SD files.
SDOPEN	Create a new SD file or prepare an existing SD file for access.
SDREAD	Read a fixed-length record from an SD file.
SDWRT	Write a fixed-length record as an SD file.

42 Terminal Device Type Codes

The terminal device type codes listed in the following table are used exclusively in the creation of TIBTAB. The columns abbreviated SS, LL, and LD represent:

SS = Screen Size (in characters)
LL = Line Length
LD = Line Depth (number of lines)

The values listed for these items can be changed within the TIBTAB by specifications of the keyword arguments FORMAT, LEN, and LINES when coding the TIB and LGCB macros.

Terminal Device Code	Terminal Type	Groupname	SS	LL	LD
BATCH	(Indicates that TIB may be used by batch)				
TTY	TELETYPE	TTY	-	80	26
TTYD	TELETYPE (DIAL)	TTYD	-	80	26
3270 L	IBM 3270 LOCAL	3270I	1920	80	24
3270 R	IBM 3270 REMOTE	3270R	1920	80	24
3275 R	IBM 3275 REMOTE	3275R	1920	80	24
3278 L	IBM 3278 LOCAL	LOCAL		80	24
3278 R	IBM 3278 REMOTE	3270R		80	24
3279 L	IBM 3279 LOCAL	LOCAL	-	80	32
3279 R	IBM 3279 LOCAL	3270R	-	80	32
3284 L	IBM 3284 LOCAL	LOCAL	-	120	20
3284 R	IBM 3284 REMOTE	3270R	-	120	20
3286 L	IBM 3286 LOCAL	LOCAL	-	120	20
3286 P	IBM 3286 REMOTE	3270R	-	120	20
3288 L	IBM 3288 LOCAL	LOCAL	-	132	18
3288 R	IBM 3288 REMOTE	3270R	-	132	18

Index

Symbols

A

B

C