

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 MVS, or CDLOAD in VSE;
- LINK in MVS;
- XCTL in MVS;
- DELETE in MVS.

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 MVS functions is functionally equivalent to that described for the Com-plete functions. The MVS 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 STEPLIB library(s) of Com-plete in MVS.

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 asynchronously 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 *areaandlength* 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 Complete 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 MVS 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 Complete 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;
- The COLINK program is locked to a thread different from that of the calling program;
- 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 the program to be loaded is thread-locked, it must be locked to the same thread as that of the calling program. 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 MVS 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.
24	The COLOAD program is locked to a thread different from that of the calling program.

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 MVS 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 overlaid by the COXCTL program. Any arguments passed to the COXCTL program are destroyed by this overlay if they reside in the calling program.

If the COXCTL function causes the program to be loaded into the application program thread region, the thread lock number of the COXCTLed program must either be the same as that of the calling program or must not be thread-locked.

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;
- The COXCTL program is locked to a thread different from that of the calling program;
- 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 Complete 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 Complete COMPLIB load library chain.

If the table or module is thread-locked, it must be locked to the same thread as that of the calling program. 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 Complete 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;
- The item being loaded is locked to a thread different from that specified for the calling program;
- 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 Complete 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.