

# The Application Programming Interface

This chapter covers the following topics:

- Overview
  - Assembler Programs Using the MCALL Interface
  - The High Level Language Interface (HLLI)
- 

## 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 advise that all applications be converted to use the new interface as soon as possible, in order to facilitate conversion to Com-plete 5.1, a bridging mechanism has been provided which enables application programs which worked under Com-plete 4.6 to function unchanged under Com-plete 5.1. 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.
- Under ESA Capable systems, 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 | <p>YES/NO Default: NO (unless previously specified in CMOPBE).</p> <p>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.</p>   |
| COMREGA  | <p>YES/NO Default: NO (unless CMOPBE was specified previously).</p> <p>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 Complete API from COMREG and to branch directly to it. This saves the requirement to link an additional stub module with the assembled module.</p> <p>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.</p> |

## Maintaining Re-entrancy

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* in this Chapter.

### 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 default 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:

- "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
- "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
- For a Description of COMREGA refer to the section How to use the Branch Entry Interface in this Chapter.

## 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 provided with Com-plete 5.1 called TLOPUSER. This module contains entry points to resolve all of the HLLI functions which can be invoked which means that only one module must now be included as against a module for each HLLI function which was invoked which was previously the case.

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.