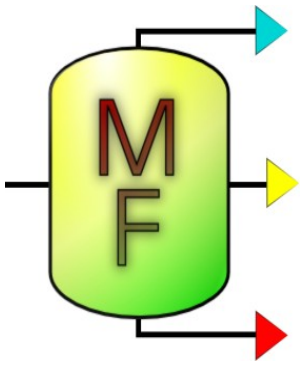
PROGRAMMER'S GUIDE FOR MULTIFLASH C++/C INTERFACE

KBC ADVANCED TECHNOLOGIES LTD

Version 6.1

June 2016



KBC Advanced Technologies ltd
42-50 Hersham Road
Walton on Thames
Surrey
KT12 1RZ
UK
T +44 (0) 1932 242 424
F +44 (0) 1932 224 214

e-mail: software@kbcat.com

DISCLAIMER

This document and the information contained within it is the copyright of Infochem/KBC Advanced Technologies Plc.

While every effort has been made to ensure that the information contained in this document is correct and that the software and data to which it relates are free from errors, no guarantee is given or implied as to their correctness or accuracy. Neither KBC Advanced Technologies Plc nor any of its employees, contractors or agents shall be liable for direct, indirect or consequential losses, damages, costs, expenses, claims or fee of any kind resulting from any deficiency, defect or error in this document, the software or the data.

CONTENTS

INTRODUCTION 2

System requirements....2

Installation instructions....2

Path settings....3

Technical support....3

Documentation....3

Contact details....4

CALLING MULTIFLASH ROUTINES FROM C++/C 5

The Multiflash DLL....5

Routines available....5

Calling Conventions....5

Argument passing....5

Data types....6

Logical data types....6

Character strings....6

Arrays....7

C header files....7

Fortran include files....7

EXAMPLES 9

Pure component properties....9

Pure component properties using a command file....11

Initialise Multiflash and do calculations....11

Initialise Multiflash and do calculations....11

INTRODUCTION

Multiflash is an advanced software package for performing complex equilibrium calculations quickly and reliably. The main utility is a multiple phase equilibrium algorithm which is interfaced to Infochem's package of thermodynamic models and a number of physical property data banks. The program also contains Infochem's Chemreact utility for performing simultaneous phase and chemical equilibrium calculations

This document describes the interface to the Multiflash DLL from Microsoft Visual C++ Version 4 or later for 32 and 64 bit applications. It should be used as a supplement to the *Multiflash Programmer's Guide* which describes in detail the Fortran API. The Multiflash DLL can also be accessed from Microsoft Visual Basic via a set of simplified interface routines that are described in the *Programmer's Guide for the Multiflash Visual Basic Interface*. Other Multiflash interfaces include a Microsoft Excel add-in (XLL), a COM module that supports the CAPE-OPEN thermodynamics interfaces and a Matlab/Simulink interface. Separate documentation is available for these interfaces.

SYSTEM REQUIREMENTS

- Operating System: Microsoft Windows 8.1, 8, 7, Vista, XP or 2000.
- A Pentium-class processor.
- Visual C++ Version 4 or higher.
- The files supplied require about 30Mb of disk space.
- A port driver is required to interact with the security key, when hardware license keys are used.

INSTALLATION INSTRUCTIONS

The Multiflash DLL is supplied as part of the standard Multiflash for Windows installation. See the *Installation Guide for Multiflash for Windows*. The version 4.4 DLL is called MFL4432.DLL. The 64 bit version, which is shipped as well is called MFL4464.DLL.

Support files for C++/C developers are supplied on the \API\CPP directory branch of the Multiflash installation.

- Manual directory

MFCInt44.pdf	this document (Adobe Acrobat PDF format).

- Examples directory

Ex1.cpp	example of evaluating pure component properties.
Ex1a.cpp	example of evaluating pure component properties using a Multiflash command file to configure the calculation.
Ex1a.mfl	Multiflash command file for above example.

Ex6.cpp	Initialise Multiflash and do calculations.
Ex6a.cpp	Initialise Multiflash and do calculations using a Multiflash command file.
Ex6a.mfl	Multiflash command file for above example.

These examples must be linked against the MFL3932.lib import library.

- Header Files directory

MFTypes.h	type definitions for variables in the Multiflash dll.
MFAPI.h	prototypes for the Multiflash dll.
ax*.h. ay*.h	translations of selected Fortran .inc files containing parameter values and dimensioning limits.
Inc2h.exe	Python script for translating Fortran .inc files containing parameter values to .h files. This program is not supported – it should be used with care and the results should always be checked.

PATH SETTINGS

It is recommended that the directory in which the Multiflash DLL is placed must be on the current search path. The installation program for the Multiflash Windows program places the dll in the installation directory, typically *\PROGRAM FILES\Infochem\MF44*. You must ensure that this directory is on the path or use the LoadLibrary() Windows API function.

To set the PATH in Windows 2000/XP use Control Panel and click on System. Select the Advanced tab and the Environment button.

TECHNICAL SUPPORT

Documentation

The C++ interface is described in this document. The *Multiflash Programmer's Guide* provides information about the low-level programming API. You should also have a copy of the *Multiflash Command Reference* that describes the Multiflash command language and the *User Guide for Multiflash for Windows* that contains background information. These documents are supplied in PDF format in the API directory in the Multiflash installation directory.

Contact details

If you need further assistance contact us at:

Infochem / KBC Advanced Technologies plc
4 The Flag Store
23 Queen Elizabeth Street
London
SE1 2LP
UK

Telephone: +44 (0)20 7357 0800
Fax: +44 (0)20 7407 3927
email: software@kbcat.com

CALLING MULTIFLASH ROUTINES FROM C++/C

THE MULTIFLASH DLL

The Multiflash DLL is created using Intel Fortran 12.1 in the Digital Visual Fortran compatibility mode. This means that current projects will compile as they previously did when the DLL was created with Digital Visual Fortran.

ROUTINES AVAILABLE

All the routines defined in the *Multiflash Programmer's Guide* are exported from the DLL. The language equivalents for calls to routines are shown in the table below

Language	**Call with return value**	**Call with no return value**
Fortran	FUNCTION	SUBROUTINE
C/Visual C++	function	(void) function
Visual Basic	Function	Sub

CALLING CONVENTIONS

To call a routine in the Multiflash DLL it should be declared with the **__stdcall** modifier in the C header file in Microsoft Windows 32bit. This modifier makes the C compiler use the correct calling and naming convention for the Fortran DLL.

Arguments are pushed onto the stack from left to right and the called function pops these arguments from the stack before it returns.

The __stdcall naming convention precedes the function name with an underscore and follows it with a @ symbol and the size of the function's argument list in bytes. For example, the Multiflash routine AYPINI is declared in Fortran as

```
SUBROUTINE AYPINI( NERR, MXERR, IERR )
INTEGER NERR, MXERR, IERR(*)
```

The exported name is _AYPINI@12. The required declaration is

```
void __stdcall AYPINI( long int *, long int *, long int * );
```

ARGUMENT PASSING

All (explicitly declared) arguments in Fortran are passed by reference. When calling routines in the Multiflash DLL from a C/C++ procedure these arguments must also be passed by reference, *i.e.* as pointers.

Strings arguments in Fortran are passed by reference along with the string length, which is pushed on the stack immediately after the address of the beginning of the string, *i.e.* the string length is an integer of 4 bytes (long) in 32bit and 8 bytes (long long) in 64bit mode. If the type STRLEN defined in MFTypes.h is used, the correct integer size will always be used.

DATA TYPES

The following table lists equivalent data types to the Fortran types in the first column.

Fortran data type	**type defined in MFAPI.h**	**C data type**	**VB6 data type**
INTEGER INTEGER*4	INTEGER	int, long	Long
REAL REAL*4	REAL4	float	Single
DOUBLEPRECISION REAL*8	REAL8	double	Double
LOGICAL	LOGICAL	long (see below)	Long (see below)
CHARACTER*(*)		See section below	

LOGICAL DATA TYPES

A Fortran LOGICAL is stored as a 4 byte indicator value (0=false, other values to zero (-1 in the case of Intel Compiler). The constants Fort_TRUE and Fort_FALSE are defined in MFTypes.h. Use the MFTypes.h LOGICAL type where a Fortran LOGICAL is required.

CHARACTER STRINGS

Strings in Fortran are passed by reference with a hidden length argument. The hidden length argument consists of an unsigned 4 byte integer (or 8byte for 64bit DLL), passed by value, immediately following the address of the character string.

For example, the Fortran routine

```
SUBROUTINE EX( K1, K2 )
CHARACTER*(*) K1, K2
```

must be declared as

```
void __stdcall EX(char *, unsigned int, char *, unsigned int )
```

Fortran strings are not null-terminated. If the value assigned to a Fortran string has fewer characters than the declared length the string will be blank padded. Fortran strings in the argument list of a routine may be declared as variable length: CHARACTER*(*) or fixed length: CHARACTER*n where n is an integer. In either case

the actual string length must be specified in the hidden length argument when the routine is called and must be passed by value. The type STRLEN is defined in MFTypes.h for this purpose.

ARRAYS

Fortran and C arrays differ in two ways:

- The default value of the lower array bound is different. By default, Fortran indexes the first element of an array as 1 whereas in C the index is 0. When dealing with default Fortran arrays in C the subscripts should therefore be one lower. Fortran arrays can also be declared with explicit lower and upper bounds so it is essential to examine the declaration before deciding how to address an array.

- In arrays of more than one dimension Fortran array elements are ordered in column-major order, *i.e.* the left-most index varies the fastest, whilst C varies the right-most index the fastest (row-major order).

For example, a Fortran array declared as X(2,2) has four elements ordered as follows: X(1,1), X(2,1), X(1,2), X(2,2). The C array declared X[2][2] addresses these elements as: X[0][0], X[0][1], X[1][0], X[1][1]. The table below shows equivalences for array declarations in different languages.

Language	**Array declaration**	**Array reference from Fortran**
Fortran	*type* X(i, k)	X(i, k)
C/Visual C++	*type* X[k][i]	X(i-1, k-1)
Visual Basic	DIM X(i, k) As *type*	X(i-1, k-1)

C HEADER FILES

The files MFTypes.h and MFAPI.h contain type definitions and function prototypes for the entry points listed in the *Multiflash Programmer's Guide*. Some other necessary header files are supplied in the \API\CPP\ directory on the installation directory.

FORTRAN INCLUDE FILES

Some of the Fortran INCLUDE files described in the *Multiflash Programmer's Guide* contain dimensioning limits and other constants that are required when calling routines in the Multiflash DLL. They are included on \API\Fortran\ directory.

A command-line utility program Inc2h.exe is provided in the \API\CPP directory. The command line arguments is the inc file to convert. It is not guaranteed to be

correct in all possible cases and it is not supported – check the output header file for consistency.

EXAMPLES

Two example programs are provided in the \API\CPP\Examples\ directory of the Multiflash installation directory.

PURE COMPONENT PROPERTIES

Ex1.cpp is a C++ version of the FORTRAN example Ex1.for for obtaining vapour pressure and saturated liquid density for water as a function of temperature.

The header files MFTypes.h and MFAPI.h are required for Multiflash applications. ayjpid.h defines the codes for pure component properties and ayjdbk.h defines the codes for databanks. The object module Ex1.obj must be linked against the Multiflash dll export library MFL4432.lib (or MFL4464.lib).

When running this example, the path for the databank files (dirnam) must correspond to the correct location on your system.

```
// ****************************************************************
//
// Copyright (C) Infochem Computer Services Ltd., 2002-2010.
// All rights reserved.
//
// Example 1: Pure component properties
//
// Revised: 4 January 2010
//
// ****************************************************************
////////////////////////////////////////////////////////////////////
//

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "mftypes.h"
#include "mfapi.h"
#include "ayjdbk.h"
#include "ayjpid.h"

int main(int argc, char* argv[])
{

  INTEGER nbank, ic[1], jbank[1], ncomp, lnodb, idt;
  STRLEN length, lenkdum, lencomp;

  long npts, i;
  double tmelt[1], tcrit[1], dum[1], vp[1], dl[1], dt, t;

  char dirnam[] = "c:\\mfl\\44";    /*databank path*/
  char kdum[] = " ";
  char comp[] = "water";    /* name of component required */
```

```
// Error reporting system

  const INTEGER MXERR = 10;
  INTEGER nerr;
  INTEGER ierr[MXERR] = {0};

  idt = 0;
  length = strlen(dirnam);

/*    Set databank path:  */

  AYFLFL( dirnam, length );

/*     Initialize pure component data software:  */

  AYPINI(&nerr, &MXERR, ierr);
  if ( nerr != 0 ) {
    printf("Error initialising pure component data system\n");
    exit(1);
  }

/*     Open INFODATA data bank    */

  jbank[0] = JDINFO;
  nbank = 1;
  lenkdum = strlen(kdum);
  i = 0;
  AYPDBO( &nbank, jbank, &i , kdum, lenkdum, &lnodb, &nerr, &MXERR, ierr );
  if ( nerr != 0 ) {
    printf("Error opening INFODATA data bank\n");
    exit(1);
  }

/*     Load pure component data  */

  lencomp = strlen(comp);
  ic[0] = 0;
  AYPDBC( &lnodb, comp, lencomp, ic, &nerr, &MXERR, ierr );
  if ( nerr != 0 ) {
    printf("Error loading data for %s\n", comp);
    exit(1);
  }

/*     Get critical temperature and melting point  */

  ncomp = 0;
  AYPCNR( &ncomp, ic, &JTCRIT, tcrit, &nerr, &MXERR, ierr );
  AYPCNR( &ncomp, ic, &JTMELT, tmelt, &nerr, &MXERR, ierr );

/*   Evaluate vapour pressure and liquid density from melting point to
     critical point    */

  printf(" Data for: %s\n", comp);
  printf("\tT/K \t\t Vap. Pres./Pa \t\t Dens/(mol/m3) \n");
  npts = 10;
  dt = (tcrit[0] - tmelt[0])/(npts-1);
  for ( i =0; i < npts ; i++ ){
    t = tmelt[0] + (i)*dt;
```

```
    AYPSVP( &ncomp, ic, &t, &Fort_FALSE, vp, dum, &nerr, &MXERR, ierr );
    AYPDNL( &ncomp, ic, &t, &idt, dl, dum, dum, &nerr, &MXERR, ierr );
    printf("\t%f\t%f\t\t%f\n", t, vp[0], dl[0] );
  }
  printf("Press Enter to finish.\n");
  gets( kdum );
  return 0;
}
```

PURE COMPONENT PROPERTIES USING A COMMAND FILE

Ex1a.cpp is a C++ version of the FORTRAN example Ex1a.for. Perfect gas enthalpy, entropy and heat capacity are tabulated for a pure substance. In contrast to the first example, Multiflash is configured using a command file EX1a.mfl that identifies the component and databank to use.

INITIALISE MULTIFLASH AND DO CALCULATIONS

The file Ex6.cpp, is variation on Ex6.for. In the C++ version the Linear Gradient surface tension is also initialised, although it is not used. In this example Multiflash is initialised, 4 components are loaded, Methane, Ethane, Heptane and Water. After this the several model are loaded and some calculations are made. Two different stream types are also set up to exemplify this feature, one with water and another without water. All the proper databank files should be in the place referred in the source file.

Please refer to the source file for more details.

INITIALISE MULTIFLASH AND DO CALCULATIONS

The file Ex6a.cpp, is a variation on Ex6a with the difference that the file “ex6.mfl” is read to initialise Multiflash.