Appendix C
Creating a User DLL

Extend Mathcad Professional's power by writing your own customized functions. Your functions will have the same advanced features as Mathcad built-in functions, such as customized error messages, interruption, and exception handling in case of overflow and divide by zero. Your functions will appear in the **Insert Function** dialog box like all built-in functions. The functions may operate on complex scalars and complex arrays and they may return complex scalars, complex arrays, and error messages.

This appendix describes how to create 32-bit DLLs for Mathcad Professional. The following sections make up this appendix:

Creating dynamically linked libraries

An overview of how to write your function and fill out the **`FUNCTIONINFO`** structure.

A sample DLL

A simple example of a user-created DLL with extensive comments. This sample can be used as a template for your own DLL.

Examining a sample DLL

A detailed examination of a simple example DLL, explaining the **`COMPLEXARRAY`** and **`COMPLEXSCALAR`** structures, error handling and function registration.

Handling arrays

Using the **`COMPLEXARRAY`** structure to handle arrays.

Allocating memory

Allocating and freeing memory.

Exception handling

How Mathcad traps the floating point exceptions.

Structure and function definitions

A reference guide to the structures and functions used.

Creating dynamically linked libraries

To create customized functions, you will first need to create source code in C or C++, then compile the source code with a 32-bit compiler. Next, you will link the object files together with the MathSoft-provided **mcaduser.lib** library to create a DLL. Finally, you will place your DLL into the **userefi** subdirectory.

Writing your DLL source code

Provided below is an overview of the steps involved in creating a DLL. Refer to the rest of this appendix for specific details on how to do each step.

Writing a DLL entry point routine

When you start Mathcad Professional, it looks in the **userefi** directory for library files with a **.dll** extension. Mathcad attempts to load all such files. During this loading process, your DLL must supply Mathcad with information about your library, including the names of the functions in the library, the types of arguments the functions take, the types of values they return, and the text of possible error messages. To supply this information, your DLL must have an entry point routine. A DLL entry point routine is called by the operating system when the DLL is loaded. Because the way to specify the DLL entry point routine is linker specific, refer to the **readme.mcd** file in the **userefi** directory for linking instructions.

Registering your function

For each function in your library, there must be a **FUNCTIONINFO** structure. The **FUNCTIONINFO** structure contains the information that Mathcad uses to register a user function. **FUNCTIONINFO** structure is an argument of **CreateUserFunction**. A call to **CreateUserFunction** inside the DLL entry point routine registers your function with Mathcad.

Writing your function

You must, of course, write a C or C++ function which implements your Mathcad user function. The parameters of your C/C++ function are pointers to the return value and to the arguments. The C/C++ function returns 0 if successful, otherwise it returns an error code. The address of the C/C++ function is passed to Mathcad inside a **FUNCTIONINFO** structure. In this way, Mathcad knows to execute your code when the function is called from a Mathcad document. Refer to the description of **MyCFunction** in the reference section at the end of this appendix.

Error Handling

C/C++ functions which return error messages require an error message table in the DLL code. A call to **CreateUserErrorMessageTable** inside the DLL entry point routine informs Mathcad of the meaning of error codes returned by the C/C++ functions from the DLL.

Compiling and linking your DLL

To create your DLL you will need a 32-bit compiler such as Microsoft Visual C++ (32-bit version), Borland C++ version 4.5, or Watcom C++32 version 10.0. Instructions on compiling and linking your DLL are given in a **readme.mcd** file located in the **userefi** directory. For more specific instructions on how to link and compile your source code, refer to the user guide provided with your compiler.

A sample DLL

To get you started writing DLLs for Mathcad we include a number of code samples. The example below is the file **multiply.c** located in the **userefi\microsoft\sources\simple** subdirectory.

The file contains a function which returns the result of multiplying an array by a scalar. This code implements the Mathcad user function *multiply*(*a*, **M**), which returns the result of an array **M** multiplied by a scalar *a*. The source code is explained in detail in later sections.

Sample code

```
#include "mcadincl.h"
#define INTERRUPTED1
#define INSUFFICIENT_MEMORY2
#define MUST_BE_REAL3
#define NUMBER_OF_ERRORS3

// tool tip error messages
// if your function never returns an error, you do not need to create this table
char * myErrorMessageTable[NUMBER_OF_ERRORS] =
{
    "interrupted",
    "insufficient memory",
    "must be real"
};

// this code executes the multiplication
// see the information on MyCFunction to find out more
LRESULT MultiplyRealArrayByRealScalar(
    COMPLEXARRAY * const Product,
    const COMPLEXSCALAR * const Scalar,
    const COMPLEXARRAY * const Array )
{
    unsigned int row, col;
    // check that the scalar argument is real
    if ( scalar->imag != 0.0
    )
```

```
        // if not, display "must be real" under scalar argument
        return MAKELRESULT( MUST_BE_REAL, 1 );

    // check that the array argument is real
    if ( Array->hImag != NULL )

        // if not, display "must be real" under array argument
        return MAKELRESULT( MUST_BE_REAL, 2 );

    // allocate memory for the product
    if( !MathcadArrayAllocate( Product, Array-rows,
    Array-cols,

        // allocate memory for the real part
        TRUE,

        // do not allocate memory for the imaginary part
        FALSE ))

        // if allocation is not successful, return with the appropriate error code
        return  INSUFFICIENT_MEMORY;

    // if all is well so far, perform the multiplication
    for ( col = 0; col < Product-> cols; col++ )
    {
        // check that a user has not tried to interrupt the calculation
        if ( isUserInterrupted() )
        {
            // if user has interrupted, free the allocated memory
            MathcadArrayFree( Product );

            // and return with an appropriate error code
            return INTERRUPTED;
        }
        for ( row = 0; row < Product-> rows; row++ )
            Product->hReal[col][row] =
                Scalar-> real*Array-> hReal[col][row];
    }
    // normal return
    return 0;
}
// fill out a FunctionInfo structure with
// the information needed for registering the function with Mathcad
FUNCTIONINFO multiply =
{
// name by which Mathcad will recognize the function
"multiply",

// description of "multiply" parameters for the Insert Function dialog box
"a,M",
```

```
// description of the function for the Insert Function dialog box
"returns the product of real scalar a and real array M",

// pointer to the executable code
// i.e. code that should be executed when a user types in "multiply(a,M)="
(LPCFUNCTION)MultiplyRealArrayByRealScalar;

// multiply(a, M) returns a complex array
COMPLEX_ARRAY,

// multiply(a, M) takes on two arguments
2,

// the first is a complex scalar, the second a complex array
{ COMPLEX_SCALAR, COMPLEX_ARRAY}
};

// all Mathcad DLLs must have a DLL entry point code
// the _CRT_INIT function is needed if you are using Microsoft's 32-bit compiler
BOOL WINAPI _CRT_INIT(HINSTANCE hinstDLL, DWORD dwReason, LPVOID
lpReserved);
BOOL WINAPI DllEntryPoint (HINSTANCE hDLL, DWORD dwReason, LPVOID
lpReserved)
{
    switch (dwReason)
    {
        case DLL_PROCESS_ATTACH:
            // DLL is attaching to the address space of the current process.
            // the next two lines are Microsoft-specific
            if (!_CRT_INIT(hDLL, dwReason, lpReserved))
                return FALSE;

            // register the error message table
            // if your function never returns an error,
            // you don't need to register an error message table
            if ( CreateUserErrorMessageTable( hDLL,
                NUMBER_OF_ERRORS, myErrorMessageTable ) )
                // and if the errors register OK, register the user function
                CreateUserFunction( hDLL, &multiply );

            break;
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:

            // the next two lines are Microsoft-specific
            if (!_CRT_INIT(hDLL, dwReason, lpReserved))
                return FALSE;
            break;
    }
```

```
    return TRUE;
}
#undef INTERRUPTED
#undef INSUFFICIENT_MEMORY
#undef MUST_BE_REAL
#undef NUMBER_OF_ERRORS
```

Compiling and linking the sample DLL

If you are using a Microsoft 32-bit compiler you can compile this file with the following command

```
cl -c -I..\..\include -DWIN32 multiply.c
```

This creates an object file **MULTIPLY.OBJ**. You can use the following command to link **MULTIPLY.OBJ** with the appropriate library and place **MULTIPLY.DLL** in the **userefi** directory.

```
link -out:..\..\..\multiply.dll -dll
-entry:"DllEntryPoint" multiply.obj
..\..\lib\mcaduser.lib
```

Check to make sure the **MULTIPLY.DLL** file is in the **userefi** subdirectory. Start Mathcad and verify that *multiply* appears in the Insert Function dialog box. You are now ready to use *multiply* in Mathcad.

Examining a sample DLL

This section will examine the source code of the simple example in the previous section. Refer to the code in the sample DLL.

MyCFunction

The heart of the program is **MyCFunction**, called **MultiplyRealArrayByRealScalar** function. It performs the actual multiplication. When the user types **multiply(a,M)=**, Mathcad executes the **MultiplyRealArrayByRealScalar** routine. The value of *a* is passed to the **MultiplyRealArrayByRealScalar** function in the **Scalar** argument. The value of **M** is passed in the **Array** argument. A pointer to the return value **Product** is the first argument of the **MultiplyRealArrayByRealScalar** function.

COMPLEXSCALAR structure

The scalar value *a* is passed to the **MultiplyRealArrayByRealScalar** function in a ***COMPLEXSCALAR*** structure. The structure has two members, *real* and *imag*. The real part of *a* is stored in **Scalar-> real**, the imaginary part in **Scalar-> imag**.

COMPLEXARRAY structure

The array value **M** is passed to the **MultiplyRealArrayByRealScalar** function in a ***COMPLEXARRAY*** structure. The **COMPLEXARRAY** structure has four members, *rows*, *cols*, *hReal*, and *hImag*. The number of rows in **M** is found in **Array-> rows**, the number of columns is found in **Array-> cols**. The real part of the element $\mathbf{M}_{row,col}$ is found in **Array-> hReal[col][row]** and the imaginary part in **Array-> hImag[col][row]**. If no element of **M** has an imaginary part, **Array-> hImag** is equal to NULL. If all elements of **M** are purely imaginary, **Array-> hReal** is equal to NULL.

The result of the multiplication of **M** by *a* is stored by the program in the **COMPLEX-ARRAY** structure pointed to by the argument **Product**. Note the memory for the multiplication result is allocated inside the **MultiplyRealArrayByRealScalar** function with a call to the **MathcadArrayAllocate** function.

Error Messages

If the multiplication was successful, **MultiplyRealArrayByRealScalar** stores the result in the **COMPLEXARRAY** structure pointed to by the argument **Product** and returns 0. In the case of an error, its return value has two components. One is the error code and the other is the location in which to display the error message.

Look at the error message table from the top of the file:

```
char * myErrorMessageTable[NUMBER_OF_ERRORS] =
{
   "interrupted",
   "insufficient memory",
   "must be real"
};
```

The function **MultiplyRealArrayByRealScalar** returns **MAKELRESULT(3,1)** to display string number 3, "must be real," under the first argument of **multiply(*a*,M)**. If **MathcadArrayAllocate** is unable to allocate memory, **MultiplyRealArrayB-yRealScalar** returns 2 to display string number 2, "insufficient memory," under the function name.

As shown in the sample DLL code, the following steps are involved in producing an error message:

- creation of an array of error message strings.
- registering the error message strings with Mathcad via a call to **CreateUserEr-rorMessageTable**. This call is made within the DLL entry point routine.
- returning an appropriate error code from the user function.

DLL entry point function

The DLL entry point is called by the operating system when the DLL is loaded. Mathcad requires that you register your user functions and your error message table while the DLL is being loaded. Note how this is done in the following code lines.

```
BOOL WINAPI DllEntryPoint (HINSTANCE hDLL, DWORD dwReason, LPVOID
lpReserved)
{
   switch (dwReason)
   {
      case DLL_PROCESS_ATTACH:

         if ( CreateUserErrorMessageTable( hDLL,
            NUMBER_OF_ERRORS, myErrorMessageTable ) )
            // if the errors register OK, register user function
            CreateUserFunction( hDLL, &multiply );

         break;
      case DLL_THREAD_ATTACH:
      case DLL_THREAD_DETACH:
      case DLL_PROCESS_DETACH:
         break;
   }
   return TRUE;

}
```

CreateUserErrorMessageTable registers the error messages. **CreateUserFunction** registers the function. You can register only one error message table per DLL, but you can register more than one function per DLL.

FUNCTIONINFO structure

The **FUNCTIONINFO** structure, *multiply*, is used for registering the DLL function with Mathcad. It contains information about the name by which Mathcad recognizes the function, the description of the function parameters, its arguments, its return value, and the pointer to the code which executes the function.

```
FUNCTIONINFO multiply =
{
   "multiply",
   "a,M",
   "returns the product of real scalar a and real array M",
   (LPCFUNCTION)MultiplyRealArrayByRealScalar;
   COMPLEX_ARRAY,
   2,
   {COMPLEX_SCALAR, COMPLEX_ARRAY}
};
```

Precision

Data is passed between Mathcad and **MyCFunction** in double precision. Use the appropriate conversion inside **MyCFunction** for different data types.

mcadincl.h

MathSoft provides the **mcadincl.h** include file. This file contains the prototypes for the following functions: **CreateUserFunction**, **CreateUserErrorMessageTa-**

ble, **MathcadAllocate**, **MathcadFree**, **MathcadArrayAllocate**, **MyCFunction**, **MathcadArrayFree**, **isUserInterrupted**. **mcadincl.h** also includes the type definitions for the structures **COMPLEXSCALAR**, **COMPLEXARRAY**, and **FUNCTIONINFO**.

Handling arrays

If your function takes an array as its argument or returns an array, refer to the **COMPLEXARRAY** structure description in "Structure and function definitions" on page 652. Note that the arrays are two-dimensional and the structure contains information about the size of the arrays and the pointers to the real and imaginary parts of the array. Refer to the next section "Allocating Memory" below for how to allocate memory inside **COMPLEXARRAY** structures.

Allocating memory

The first argument of **MyCFunction** is a pointer to a return value. If it points to a **COMPLEXARRAY** structure, you will need to allocate memory for the members of this structure using **MathcadArrayAllocate**. If **MyCFunction** is returning an error, free the memory allocated for the return value using **MathcadArrayFree**. In the case of an error-free return, do not free the memory allocated for the return value.

Use the **MathcadAllocate** and **MathcadFree** functions to allocate and free memory inside **MyCFunction**.

Exception handling

Mathcad traps the following floating point exceptions; overflow, divide by zero, and invalid operation. In the case of these exceptions, Mathcad will display a floating point error message under the function. Mathcad will also free all the memory allocated inside **MyCFunction** with **MathcadArrayAllocate** and **MathcadAllocate**.

Structure and function definitions

This section describes in more detail the structures and functions used in creating your own dynamically linked library.

The COMPLEXSCALAR Structure

```
typedef struct tagCOMPLEXSCALAR {
   double real;
   double imag;
} COMPLEXSCALAR;
```

The **COMPLEXSCALAR** structure is used to pass scalar data between Mathcad and a user DLL. The real part of a scalar is stored in the *real* member of a **COMPLEXSCALAR**, and the imaginary in the *imag* member.

Member	Description
real	Contains the real part of a scalar.
imag	Contains the imaginary part of a scalar.

The COMPLEXARRAY Structure

```
typedef struct tagCOMPLEXARRAY {
   unsigned int rows;
   unsigned int cols;
   double **hReal;
   double **hImag;
} COMPLEXARRAY;
```

The **COMPLEXARRAY** structure is used to pass array data between Mathcad and a user DLL. It contains the information about the size of the array and whether any of the elements in the array has an imaginary or a real component.

Member	Description
rows	Number of rows in the array.
cols	Number of columns in the array.
hReal	Points to the real part of a complex array *hReal[i][j]* contains the element in the *i*th column and the *j*th row of the array. *hReal* is equal to NULL if the array has no real component.
hImag	Points to the imaginary part of a complex array *hImag[i][j*], contains the element in the *i*th column and the *j*th row of the array. *hImag* equals NULL if the array has no imaginary component.

Comments

hReal and *hImag* members of the argument array are indexed as two-dimensional array of the range [0 .. *cols* – 1][0 .. *rows* – 1].

The FUNCTIONINFO Structure

```
typedef struct tagFUNCTIONINFO{
   char * lpstrName;
   char * lpstrParameters;
   char * lpstrDescription;
   LPCFUNCTION lpfnMyCFunction;
   long unsigned int returnType;
   unsigned int nArgs;
   long unsigned int argType[MAX_ARGS];
} FUNCTIONINFO;
```

The **FUNCTIONINFO** structure contains the information that Mathcad uses to register a user function. Refer below for each member and its description.

Member	Description
lpstrName	Points to a NULL-terminated string that specifies the name of the user function.
lpstrParameters	Points to a NULL-terminated string that specifies the parameters of the user function. The string is used by the Insert Function dialog box.
lpstrDescription	Points to a NULL-terminated string that specifies the function description for the Insert Function dialog box.
lpfnMyCFunction	Pointer to the code that executes the user function.
returnType	Specifies the type of value returned by the function. The values are **COMPLEX_ARRAY** or **COMPLEX_SCALAR**.
nArgs	Specifies the number of arguments expected by the function. Must be between 1 and **MAX_ARGS**.
argType	Specifies an array of long unsigned integers containing input parameter types.

CreateUserFunction

```
const void * CreateUserFunction(hDLL, functionInfo)
HINSTANCE hDLL;
FUNCTIONINFO * functionInfo;
```

CreateUserFunction is called when the DLL is attaching to the address space of the current process in order to register the user function with Mathcad.

Parameter	Description
hDLL	Handle of the DLL supplied by the DLL entry point routine.

functionInfo	Points to the **FUNCTIONINFO** structure that contains information about the function.
	The **FUNCTIONINFO** structure has the following form:
	`typedef struct tagFUNCTIONINFO{`
	`  char * lpstrName;`
	`  char * lpstrParameters;`
	`  char * lpstrDescription;`
	`  LPCFUNCTION lpfnMyCFunction;`
	`  long unsigned int returnType;`
	`  unsigned int nArgs;`
	`  long unsigned int argType[MAX_ARGS];`
	`} FUNCTIONINFO;`

Return value

The return value is a non-NULL handle if the registration is successful. Otherwise, it is NULL.

CreateUserErrorMessageTable

```
BOOL CreateUserErrorMessageTable(hDLL,n,ErrorMessageTable)
HINSTANCE hDLL;
unsigned int n;
char * ErrorMessageTable[ ];
```

CreateUserErrorMessageTable is called when the DLL is attaching to the address space of the current process in order to register the user error message table with Mathcad.

Parameter	Description
hDLL	Handle of the DLL supplied by the DLL entry point routine.
n	Number of error messages in the table.
ErrorMessageTable	An array of *n* strings with the text of the error messages.

Return value

The return value is TRUE if the registration is successful. Otherwise, it is FALSE.

MathcadAllocate

```
char * MathcadAllocate(size)
unsigned int size;
```

Should be used to allocate memory inside the **MyCFunction**. Allocates a memory block of a given size (in bytes) of memory.

Parameter	Description
size	Size (in bytes) of memory block to allocate. Should be non-zero.

Return value

Returns a pointer to the storage space. To get a pointer to a type other than char, use a type cast on the return value. Returns NULL if the allocation failed or if size is equal to 0.

MathcadFree

```
void MathcadFree(address)
char * address;
```

Should be used to free memory allocated with **MathcadAllocate**. The argument address points to the memory previously allocated with **MathcadAllocate**. A NULL pointer argument is ignored.

Parameter	Description
address	Address of the memory block that is to be freed.

Return value

The function does not return a value.

MathcadArrayAllocate

```
BOOL MathcadArrayAllocate(array, rows, cols, allocateReal,
allocateImaginary)
COMPLEXARRAY* const array;
unsigned int rows;
unsigned int cols;
BOOL allocateReal;
BOOL allocateImaginary;
```

Allocates memory for a **COMPLEXARRAY** of *cols* columns and *rows* rows. Sets the *hReal*, *hImag*, *rows* and *cols* members of the argument array.

Parameter	Description

array	Points to the **COMPLEXARRAY** structure that is to be filled with the information about an array.
	The **COMPLEXARRAY** structure has the following form:
	
	`typedef struct tagCOMPLEXARRAY {`
	`  unsigned int rows;`
	`  unsigned int cols;`
	`  double **hReal;`
	`  double **hImag;`
	`} COMPLEXARRAY;`

rows	Row dimension of the array that is being allocated. After a successful allocation, the *rows* member of the argument array is set to the value of *rows*.
cols	Column dimension of the array that is being allocated. After a successful allocation, the *cols* member of the argument array is set to the value of *cols*.
allocateReal	Boolean flag indicating whether a memory block should be allocated to store the real part of the array. If *allocateReal* is FALSE the function does not allocate storage for the real part of array and sets the *hReal* member to NULL.
allocateImag	Boolean flag indicating whether a memory block should be allocated to store the imaginary part of the array. If *allocateImag* is FALSE the function does not allocate storage for the imaginary part of array and sets the *hImag* member to NULL.

Return value

Returns TRUE if the allocation is successful, FALSE otherwise.

Comments

hReal and *hImag* members of the argument array are allocated as 2-dimensional array of the range [0 .. *cols* – 1][0 .. *rows* – 1].

MyCFunction

```
LRESULT MyCFunction(returnValue, argument1,...)
void * const returnValue;
const void * const argument1;
...
```

MyCFunction is the actual code which executes the user function. Mathcad arguments and a pointer to a return value are passed to this function. It puts the result of the calculation in the return value.

Parameter	Description
returnValue	Points to a **COMPLEXARRAY** or a **COMPLEXSCALAR** structure where the function result is to be stored. If you are implementing a Mathcad user function which returns a scalar, *returnValue* is a pointer to a **COMPLEXSCALAR** structure. If you are implementing a Mathcad user function that returns an array, *returnValue* points to a **COMPLEXARRAY** structure.
argument1	Points to a read-only **COMPLEXARRAY** or a **COMPLEXSCALAR** structure where the first function argument is stored. If you are implementing a Mathcad user function that has a scalar as its first argument, *argument1* is a pointer to a **COMPLEXSCALAR** structure. If you are implementing a Mathcad user function that has an array as its first argument, *argument1* points to a **COMPLEXARRAY** structure.

...	If you are implementing a Mathcad user function that has more than one argument, your **MyCFunction** will have additional arguments. The additional arguments will be pointers to the read-only **COMPLEXARRAY** or a **COMPLEXSCALAR** structures where the data for the corresponding Mathcad user function argument is stored.

Return value

MyCFunction should return 0 to indicate an error-free return. To indicate an error **MyCFunction** should return an error code in the low word of the returned **LRESULT**, and in the high word the number of the argument under which the error box should be placed. If the high word is zero the error message box is placed under the function itself. See the section on error handling to find out more about error codes.

Comments

MyCFunction is a placeholder for the library-supplied function name. You can name the function that executes your Mathcad user function anything you would like, but you must register the address of your executable code with Mathcad by setting the **lpfnMyCFunction** member of the **FUNCTIONINFO** structure.

MathcadArrayFree

```
void MathcadArrayFree(array)
COMPLEXARRAY * const array;
```

Frees memory that was allocated by the **MathcadArrayAllocate** function to the *hReal* and *hImag* members of the argument array.

Parameter	Description

array	Points to the **COMPLEXARRAY** structure that is to be filled with the information about an array.
	The **COMPLEXARRAY** structure has the following form:
	`typedef struct tagCOMPLEXARRAY {`
	`  unsigned int rows;`
	`  unsigned int cols;`
	`  double **hReal;`
	`  double **hImag;`
	`} COMPLEXARRAY;`

Return value

The function does not return a value.

isUserInterrupted

```
BOOL isUserInterrupted(void)
```

The **isUserInterrupted** function is used to check whether a user has pressed the **[Esc]** key. Include this function if you want to be able to interrupt your function like other Mathcad functions.

Parameter

The function does not take any parameters.

Return value

Returns TRUE if the **[Esc]** key has been pressed, FALSE otherwise.