MatDeck plug-ins

MatDeck is a extendible software, meaning that the user can create new custom components and functions for it.

Plug-ins are code libraries, collections of functions, compiled with c++ compiler from MatDeck script.The advantage of plug-ins is that plug-in functions are executed very fast, much faster than script in the document. Plug-in functions are in the rank of the MatDeck built in functions. In windows plug-ins are dynamic link libraries (dll files) located in plug-ins subdirectory of the application installation directory and all plug-ins are loaded during the program execution. Its functions can be used like any other built in functions by typing its name and without any special settings. Also plug-ins will be deployed with your custom applications (see Deploy Exe).

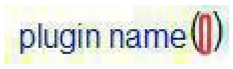
Note: This document assumes that the user has basic knowledge in computer programing (especially in c or c++ languages) and has installed our software development kit (SDK). If SDK is not installed it will be downloaded the first time when you try to build. We are recommending extracting the archive in the root directory of your main partition (for example in C:\SDK). After this you should set your SDK directory in Math Settings -> SDK Dir.

Creating simple plug-in.

Insert new canvas in the new document and add one maths object on it.

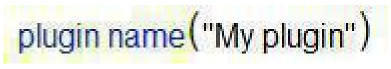
First we will define a plug-in name.

Type the "plugin name"



```
plugin name()
```

And enter the plug-in name between the quotation marks.

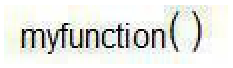


```
plugin name("My plugin")
```

Now we can create a function.

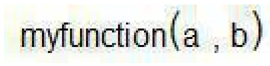
Note that function names are unique in the whole system, so you can not use the same function name twice. This is true for all keywords in the MatDeck system. Also function names are key sensitive and "myfunction" and "myFunction" are not the same.

Place another maths object or hit the Enter key. Then type the "function" keyword and type "myfunction" in the red square



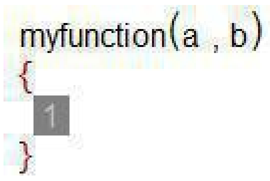
```
myfunction( )
```

Hit the right arrow key twice to place the cursor between the brackets (parent-houses).Type "a", then comma(,) to add a second argument, then type "b". In MatDeck you can add function arguments with "," and remove it with backspace.



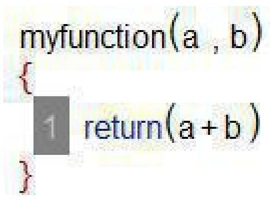
```
myfunction(a , b)
```

Now hit the right arrow key once to place the cursor right of the brackets and hit Enter for new line. Press "{" (shift+[) to form a function body.



```
myfunction(a , b)
{
  1
}
```

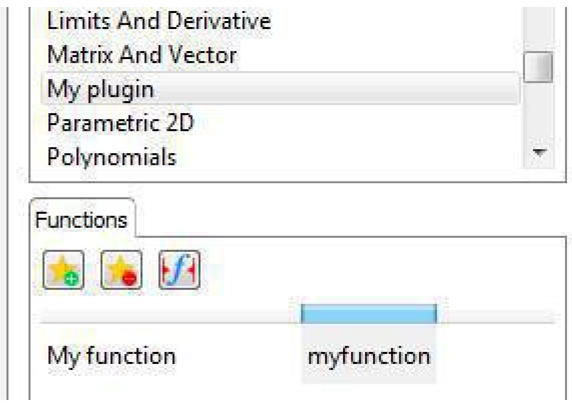
Now in the function body enter "return a+b"



```
myfunction(a , b)
{
  1  return(a+b)
}
```

You can build your plug-in now.

Click the Build Plug-in button from the Math tab. If everything is Ok your plug-in is created, installed and loaded with other plug-ins and you will see it in the functions list. See picture below.



Installing and uninstalling plug-ins

Build process will generate two files, PluginName.h and PluginName.dll (PluginName refers to the name from the first step above), in MatDeck SDK directory.

Installing is as simple as copying these two files from SDK directory to the plugins subdirectory of the MatDeck installation directory. This process is done automatic by MatDeck, but sometimes installation directory is protected from changing (changing requires administrator privileges) and the installation process will fail.

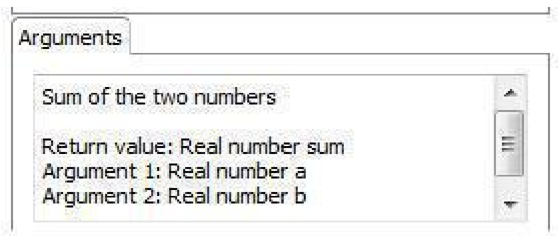
You can use two strategies in this case:

1. Save your files and close MatDeck. Copy PluginName.h and PluginName.dll from SDK directory to MatDeckInstallationDirectory/plugins manually and start MatDeck again.
2. Save your files and close MatDeck. Run MatDeck as administrator (right click on MatDeck icon and select Run as administrator). Then build plug-in again.

Uninstalling the plug-in is as simple as deleting two plug-in files from plugins subdirectory. Before deleting files MatDeck should be closed.

Creating helper functions for you function

Every function can be supported by two helper functions. One will give the function name (see My function in the picture above) and the other will give the function description in the MatDeck tool-bar and tool tips. See picture below.



Sum of the two numbers

Return value: Real number sum
Argument 1: Real number a
Argument 2: Real number b

You can create these in the manner described for myfunction. Note both functions are without arguments and both names should be exactly the same as the function name plus Name (capital N) in the first case and Description (capital D) in the second case (myfunctionName() and myfunctionDescription())

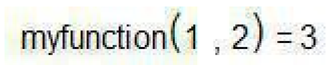
The first function should return string for the function name and the second should return vector for the function description. See code example at the end of this document. Vector size in the second function should be 2 + number of function arguments, so the first row will be function description, second is description of the return value, and every next row will be description for one argument respectively. If function has no arguments vector size should be 2 rows.

Other functions

You can define as many functions in one plug-in file as you wish. By default all functions will be public functions (user will see it in the Functions list). If you wish to keep some function private in your plug-in, name it with Private at the end (capital P). See code example below.
Classes defined or included in the plug-in code are private.

Testing

When you are done with the plug-in functions implementation you can test it by adding new maths object and typing function name, enter the function arguments and move cursor to the right side of the function call and hit equal ("=")



$\text{myfunction}(1\ ,\ 2) = 3$

Note if you are calling a function from the same document MatDeck will use function code from that document. So if you change the function body and evaluate the document (ctrl + E) you will get a new result. Unlike that if you call function from the new document MatDeck will use your plug-in function from the last plug-in build.

After you are done with testing do not forget to delete tests and build the final version of your plug-in.

Example plug-in code. You can try to build plug-in from this source

```
plugin name("My plugin")

myfunction(a , b)
{
    return(myfnPrivate(a , b))
}
myfunctionName( )
{
    return("My function")
}
myfunctionDescription( )
{
    return([ "Sum of the two numbers"
             "Real number sum"
             "Real number a"
             "Real number b" ])
}
```

```
myfnPrivate(a , b)
{
    return(a + b)
}
```