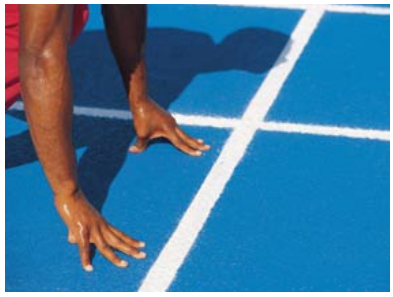
maXbox



maXbox Starter 44

Work with maXbox IDE Improvements II

1.1 Command with Macros

Today we step through optimize your maXbox. This tool is great for fast coding but also provides a mechanism for extending your functions and quality with checks and tests.

You can set the macros like #host: in your header or elsewhere in a comment line, but not two or more on the same line when it expands with content:

Try: `369_macro_demo.txt`

```
{*************************************************************
 * Project   : Macro Demo
 * App Name  : #file:369_macro_demo.txt
 * Purpose   : Demonstrates functions of macros in header
 * Date      : 21/09/2010  - 14:56 - #date:01.06.2013 16:38:20
 *             #path E:\maxbox\maxbox3\examples\
 *             #file 369_macro_demo.txt
 *             #perf-50:0:4.484
 * History   : translate/implement June 2013, #name@max
 *           : Sys demo for mX3, enhanced macros, #locs:149
 *************************************************************}
```

As you may know there's no simple solution put a macro because the step of the preprocessor has to be the first to expand. So a macro is not part of the source code and the best way is to set a macro always as a one liner or somewhere else in comment.

All macros are marked with yellow. One of my favour is #locs: means lines of code metric and you get always the certainty if something has changed by the numbers of line. So the editor has a programmatic macro system which allows the pre compiler to be extended by user code I would say user tags. Below an internal extract from the help file All Functions List: `maxbox_functions_all.pdf`

Some macros produce simple combinations of one liner tags but at least they replace the content by reference in contrary to templates which just copy a content by value.
After the end. of the code section you can set a macro without comment signs so you can enlarge your documentation with version and time information like:

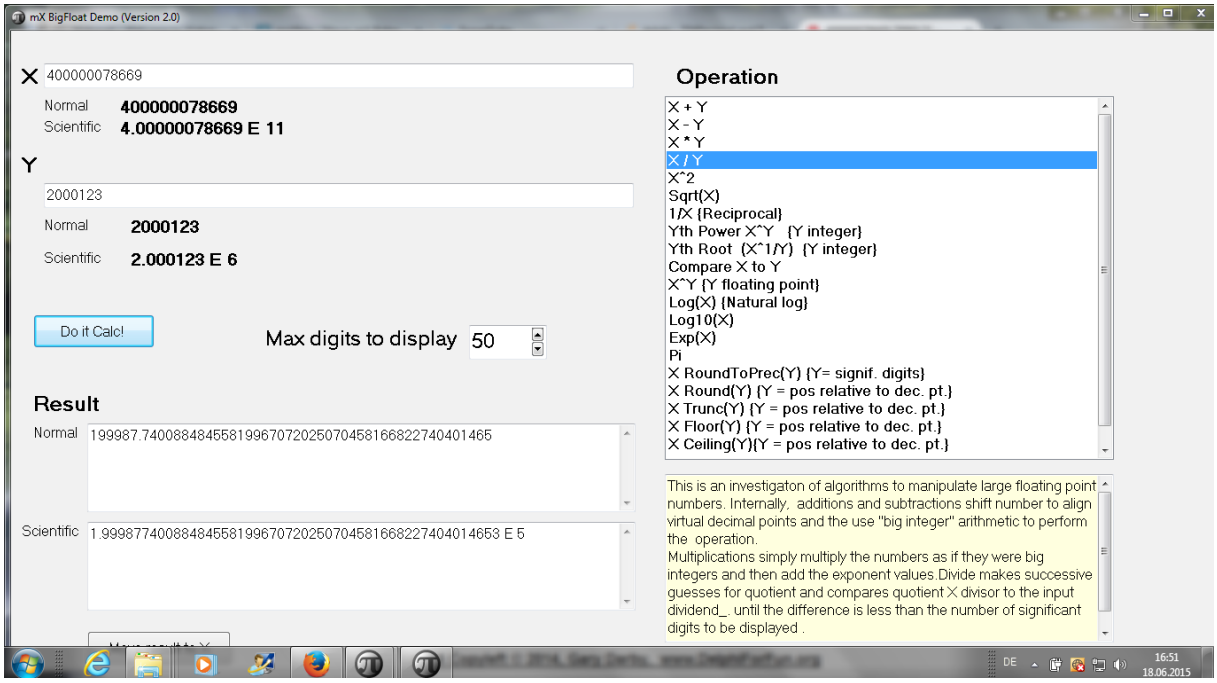
```
#tech:perf: 0:0:1.151 threads: 4 192.168.56.1 12:06:17 4.2.2.95
```

Or you put a macro behind a code-line, so this is not the end of the line, a second time test is

```
maxcalcF('400000078669 / 2000123')
//#perf>0:0:1.163
```

and we get: 199987.740088485

maXbox3 `568_U_BigFloatTestscript2.pas` Compiled done: 6/18/2015



29×37×127^(-1)×179×15749^(-1)×2082607

```
maxcalcF('29*37*(127^-1)*179*(15749^-1)*2082607');
```

>> 199987.740088485

```
maxcalcF('29*37*(127^-1)*179*(15749^-1)*2082607');
```

```
//#tech>perf: 0:0:1.190 threads: 6 192.168.56.1 12:49:55 4.2.2.95
//>>> 199987.740088485
```

1.2 Extend with DLL

A DLL is a library, short for Dynamic Link Library, a library of executable functions or data that can be used by a Windows or Linux application. This is how we declare a function we want to use from a DLL:

```
Function OpenProcess2(dwDesiredAccess: DWORD;
         bInheritHandle: BOOL; dwProcessId:DWORD): THandle;
               External 'OpenProcess@kernel32.dll stdcall';
```

Suppose you want to use the function `OpenProcess` of the '`kernel32.dll`'. All you have to do is to declare above statement and you get access to the kernel! With external you made these functions available to callers external to the DLL, so we must export them or at least say the function we use is from `External`.
This means also to use the modifier `stdcall` because this is a C convention. The function name `OpenProcess2` is different from the original name `OpenProcess`! This is an alias to prevent name conflicts or name it you like because you do have conventions you are free to rename the function in your script.
Or a function is missing, for example `gettickcount64`

No problem you search for the DLL and include it:

There's a template `dll`, write the word `dll` maybe at the beginning or below of your code and press `<Ctrl> j` and it gets expanded to:

```
function  MyGetTickCount: Longint;
    external 'GetTickCount@kernel32.dll stdcall';
```

Now we can change it:

```
function  MyGetTickCount64: Longint;
    external 'GetTickCount64@kernel32.dll stdcall';
```

What about big integers in a DLL? For example you want to compute fact(70), your calculator shows:

fact(70) = 1.19785716699699e+100 or **maxcalcF**('70!')

1.19785716699699E100

or even more

1.19785716699698917960727837216890987364589381425464258 57...
× 10^100
So find the right DLL as `BigInt` or `BigDecimal` but the maximum range on Pascal or C depends on your operating system types, means nowadays an int64 range is big but maXbox has some `BigInt` libs already.

1.3 Alias Naming

Most of the functions can have a second name for example:

Inc() and Inc1() //one parameter or two parameter

So if you cant run a function try the second one, for ex. Voice()-Voice2(), inc() - inc1(), Rect – Rect2, Abs - AbsInt and many more.

Now that the "signed" functions with 1, 2, 3 and more at the end of the function name is a workaround to provide the overload which maXbox doesn't support.

Another way is to use a type extended, but the limitation is precision like

```
Writeln(FloatToStr(Fact(70)))
```

it only shows 1.2E+0100 or 1.19785716699698918E100

With a second one there's more parameters possible:

```
Function FloatToStr2(Value: Extended; Format: TFloatFormat;
Precision, Digits: Integer; FormatSettings: TFormatSettings):
string;
```

```
Writeln(FloatToStr2(Fact(70),ffcurrency,18,6,formatSettings))
```

By the way with a BigInt Library you'll see the full range of Fact(70):

11978571669969891796072783721987892755536628009582789845319680000000000000000

All examples can be found online:

```
maxbox4\examples\161_bigint_class_maxprove3.txt
```

http://www.softwareschule.ch/examples/161_bigint_class_maxprove3.txt

The call respectively the calculation goes like this:

```
function GetBigIntFact(aval: byte): string;
//call of unit mybigint
var mbRes: TmyBigInt;  i: integer;
begin
  mbRes:= TMyBigInt.Create(1);
  try
    //multiplication of factor
    for i:= 1  to aval do
      mbRes.Multiply1(mbres, i);
```

```
    Result:= mbRes.ToString;
  finally
    //FreeAndNil(mbResult);
    mbRes.Free;
    mbRes:= Nil;
  end;
end;
```

1.4 Console Capture DOS

I'm trying to move a part of SysTools to Win64. There is a certain class `TStDecimal` which is a fixed-point value with a total of 38 significant digits. The class itself uses a lot of ASM code.

```
function BigDecimal(aone: float; atwo: integer): string;
begin
  with TStDecimal.create do begin
    try
      //assignfromint(aone)
      assignfromfloat(aone) //2
      RaiseToPower(atwo)  //23
      result:= asstring
    finally
      free
    end;
  end;
end;
```

But then I want to test some Shell Functions on a DOS Shell or command line output. The code below allows to perform a command in a DOS Shell and capture it's output to the maXbox console. The captured output is sent “real-time” to the Memo2 parameter as console output in maXbox:

```
    srlist:= TStringlist.create;
      ConsoleCapture('C:\', 'cmd.exe', '/c dir *.*',srlist);
      writeln(srlist.text)
    srlist.Free;
```

But you can redirect the output `srlist.text` anywhere you want.
For example you can capture the output of a DOS console and input into a textbox, or you want to capture the command start of demo app and input into your app that will do further things.

```
ConsoleCapture('C:\', 'cmd.exe', '/c ipconfig',srlist);
ConsoleCapture('C:\', 'cmd.exe', '/c ping 127.0.0.1',srlist);
```

☝ It is important to note that some special events like `/c java -version` must be captured with different parameters like /k or in combination.

Here's the solution with `GetDosOutput()`:

```
writeln('GetDosOut: '+GetDosOutput('java -version','c:\'));
```

or like the man-pages in Linux

```
writeln('GetDosOut: '+GetDosOutput('help dir','c:\'));
```

1.5 Byte Code Performance

Intermediate representations such as byte-code may be output by programming language implementations to ease interpretation, or it may be used to reduce hardware and operating system dependence by allowing the same code to run on different platforms. So you can share your code as source in a normal text-file (*.txt) or as bytecocde (*.psb) to gain speed or obfuscation.

In some cases, you may want to export or deliver a script with its byte-code to store on removable media or to use on a different computer without the source as a text-file. This is how you can do that:

1. You open a script and compile it before.
2. you go to `/Options/Save Bytecode/` and the console writes:

```
-----PS-BYTECODE (PSB) mX4-----13:48:38 -----BYTECODE
saved as:
C:\maXbook\maxbox3\mX3999\maxbox3\examples\287_eventhandl
ing2_primewordcount.psb
----IFPS#
```

3. you load the byte-code by `/Options/Load Bytecode`...

```
IFPS#
### mX3 byte code executed: 10.06.2015 13:53:20 Runtime:
0:0:1.577 Memoryload: 60% use ByteCode Success Message
of: 287_eventhandling2_primewordcount.psb
```

4. When testing is finished you send the byte-code to your client

But there are some restrictions to this procedure: You should avoid self referencing commands like `maxform1.color` or `memo1.text` in your byte-code. Also reflection calls, unsafe type casts or runtime type information can fail.
So during development, we may want to create code for our own purposes and then implicitly share them to test, deploy or reuse. But we don't really want to go to an app- or store authority and get a signed certificate for

code signing, because that costs money and makes us dependent. If you don't want to share the source code for property or security reasons you can deliver the byte-code of the script which they can load and run in maXbox like previously said. On the other side you want that others can trust to your code so you deliver them simply the SHA1 hash of the file as a signature:

5. You open the byte-code as a normal file in the editor (see below):

6. ..\examples\287_eventhandling2_primewordcount.psb File loaded

7. you go to `/Program/Information` and copy the SHA1 of file: F01801AA105463EC6A937602AD2FA67DAA06C8D0

8. you send this number (fingerprint) to your client

9. Client loads the byte-code and compares the fingerprint to verify! By meaning signature is just a hash. Real Code Signing uses certificate associated with key pairs used to sign active content like a script or an application explained above. The storage location is called the certificate store. A certificate store often has numerous certificates, possibly issued from a number of different certification authorities.

Let's have a look at some byte-code for fun.

go to ♦-ptions♦"ave Bytecode♦ and the console writes(+++++,-+♀✔/&%[■]!& [ ],-♀■ mX#+++++$34#8438+++++♀✔/& %♦!& saved as4\maXboo5\maxbox3\mX3 (((\maxbox3\examples\ 287_eventhandling2_primewordcount.psb +++++6F,-❍

1.6 Exception Handling

A few words how to handle Exceptions within maXbox:
Prototype:

```
procedure RaiseException(Ex: TIFException; const Msg: String);
```

Description:
Raises an exception with the specified message.
Example:

```
begin
  RaiseException(erCustomError,'Your message goes here');
// The following line will not be executed because of the
exception!
  MsgBox('You will not see this.', 'mbInformation', MB_OK);
end;
```

This is a simple example of a actual script that shows how to do try except with raising a exception and doing something with the exception message.

procedure Exceptions_On_maXbox;

```
var filename,emsg:string;
begin
   filename:= '';
   try
     if filename = '' then
       RaiseException(erCustomError,
                    'Exception: File name cannot be blank');
   except
     emsg:= ExceptionToString(ExceptionType, ExceptionParam);

       //do act with the exception message i.e. email it or
       //save to a log etc

     writeln(emsg)
   end;
end;
```

☝ The `ExceptionToString()` returns a message associated with the current exception. This function with parameters should only be called from within an except section.

1.7 Config the Ini File

Test the script with **F9** / F2 or press Compile.
As you already know the object we now step through the meaning of the ini file **maxboxdef.ini**. On subsequent execution of maXbox, the ini values are read in when the form is created and written back out in the `OnClose` and other “in between” events.

In maXbox you can also start with read only mode (`Options/Save before Compile`), so nothing will be write on the disk.

//*** Definitions for maXbox mX4 ***

```
[FORM]
LAST_FILE=E:\maxbox\maxbox3\examples\140_drive_typedemo.txt
                        //history up to 10 files
FONTSIZE=14             //editor
EXTENSION=txt           //default
SCREENX=1386            //window size
SCREENY=1077
MEMHEIGHT=350           //Console Output
```

```
PRINTFONT=Courier New    //GUI Settings
LINENUMBERS=Y
EXCEPTIONLOG=Y      //store exceptions in log file - menu
                      Debug/Show Last Exceptions
EXECUTESHELL=Y      //prevents execution of ExecuteShell() or
                      ExecuteCommand() or WebScript
BOOTSCRIPT=Y        //enabling load a boot script
MACRO=Y             //put macros in your source header file
NAVIGATOR=Y         //set the navigator listbox at the right
                      side of code editor
MEMORYREPORT=Y      //shows memory report on closing maXbox
```

[WEB]

```
IPPORT=8080
//for internal webserver - menu /Options/Add Ons/WebServer2
IPHOST=192.168.1.53
ROOTCERT='filepathY' //for use of HTTPS and certificates…
SCERT='filepathY'
RSAKEY='filepathY'

APP=C:\WINDOWS\System32\calc.exe
                     //set path to an external app
MYSCRIPT=E:\mXGit39991\maxbox3\examples\330_myclock.txt
                     //start script of menu /View/MyScript
VERSIONCHECK=Y       //checks over web the version
```

The last entry VERSIONCHECK can enhance the load speed cause there's no request to HTTP and also no sense for firewalls to signal an outgoing socket call!

If you don't want execute scripts from the web you can stop this in the ini-file **maxboxdef.ini** with N.

This is seen in the following figure.

LINENUMBERS=Y

EXCEPTIONLOG=Y

EXECUTESHELL=N

MEMORYREPORT=Y

BOOTSCRIPT=Y

The ini file format is still popular; many configuration files (such as Desktop or Persistence settings file) are in this format. This format is especially useful in cross-platform applications, where you can't always count on a system Registry for storing configuration information. I never was a friend of the Registry so you can also start maXbox from a stick.

```
if DownloadURLToFile('http://www.softwareschule.ch/maxboxnews.htm',
             exepath+'localwebstore.txt') then
                openDoc(exepath+'localwebstore.txt');
```

1.8 The Log Files

There are 2 log files a runtime log and an exception log:
using Logfile: `maxboxlog.log` , Exceptionlogfile: `maxboxerrorlog.txt`

```
New Session Exe Start C:\maXbox\tested
>>>> Start Exe: maXbox4.exe v4.0.2.80 2016-02-03 14:37:18
>>>> Start [RAM  monitor] : Total=2147483647,
Avail=2147483647, Load=30% ; [Disk monitor] : Available to
user=671317413888, Total on disk=972076589056, Free total on
disk=671317413888 ; 2016-02-03 14:37:18

>>>> Start Script: C:\Program Files
(x86)\Import\maxbox4\examples\640_weather_cockpit6_1.TXT
2016-02-03 14:37:33 From Host: maXbox4.exe of
C:\maXbox\maxbox3\work2015\Sparx\
>>>> Stop Script: 640_weather_cockpit6_1.TXT
[RAM  monitor] : (2147483647, 2147483647, 30%) Compiled+Run
Success! Runtime: 14:37:33.267

New Session Exe Start C:\Program Files(x86)\Import\maxbox4\
examples\640_weather_cockpit6_1.TXT
>>>> Start Exe: maXbox4.exe v4.2.2.95 2016-05-19 09:15:17
>>>> Start [RAM  monitor] : Total=2147483647,
Avail=2147483647, Load=25% ; [Disk monitor] : Available to
user=675888001024, Total on disk=972076589056
```

Also possible to set report memory in script to override ini setting
procedure Set_ReportMemoryLeaksOnShutdown(abo: boolean)

1.9 Use Case Model

Also possible is to set for each script a use case diagram with the extension *.uc. They are associated together by opening the script you can open the model too.

So far so good now we 'll open our two examples:

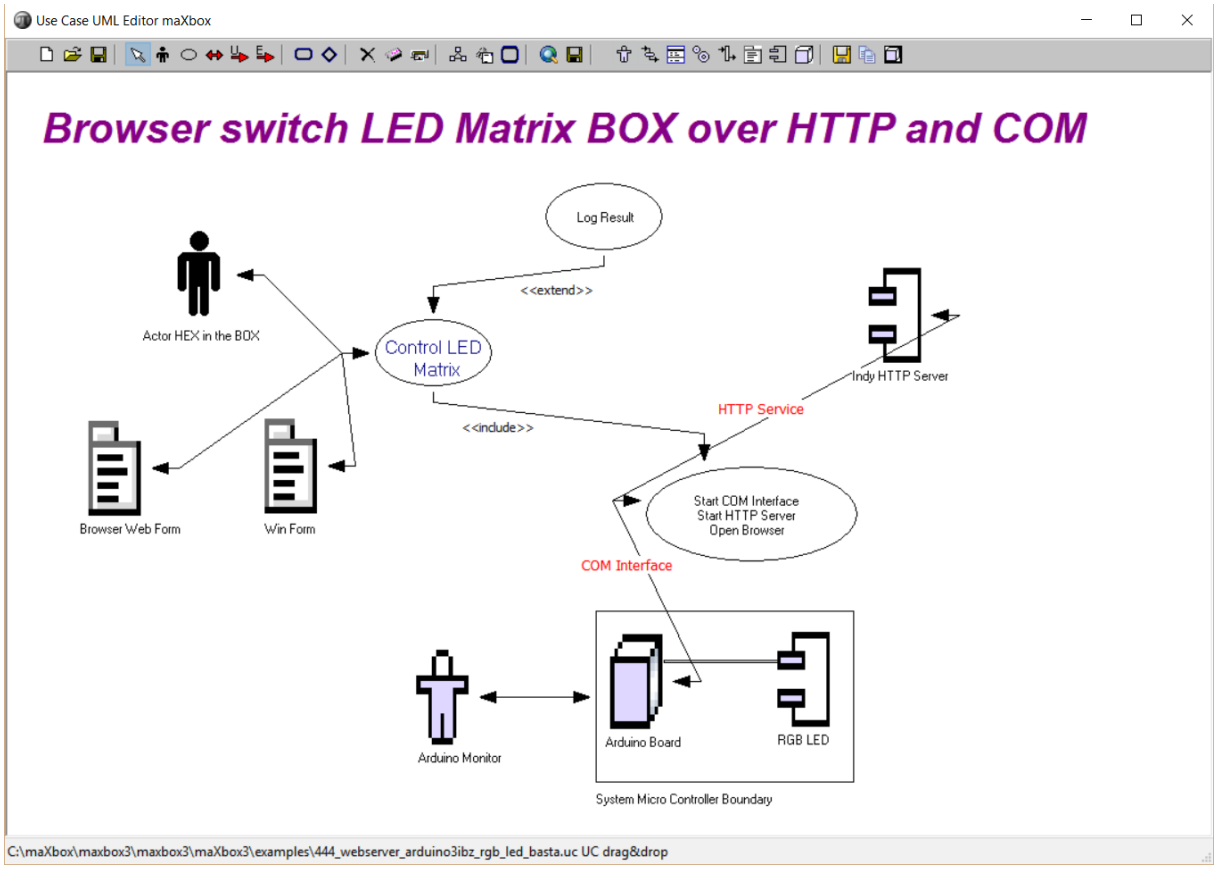
```
59_timerobject_starter2_uml_main.txt
59_timerobject_starter2_uml_form.uc
```

Among other items, during the pre-processor step the compiler is looking for compiler directives or macros and processes them as they are encountered.
After completing the tasks as directed, the compiler proceeds to its second step where it checks for syntax errors (violations of the rules of the language) and converts the source code into an object code that contains

machine language instructions, a data area, and a list of items to be resolved when he object file is linked to other object files.

At least there are two ways to install and configure your box into a directory you want. The first way is to use the unzip command-line tool or IDE, which is discussed above. That means no installation needed. Another way is to copy all the files to navigate to a folder you like, and then simply drag and drop another scripts into the /examples directory.

The only thing you need to backup is the ini file **`maxboxdef.ini`** with your history or another root files with settings that have changed, otherwise the unzip IDE overwrites it.



1.10 Open Tool API

At last we go back to the magic boot script which will be the key to modify the IDE especially with the inbuilt SynEdit API (since V3.9.8.9). What does it mean? It means you can change or rebuild your IDE not just by fixed options or settings but also in a programmatic way in your boot script without compilation!
Imagine you want to set a vertical red line on the gutter to the left:

```
//memo1.Gutter.BorderColor:= clred;          //---> reflection to box!
//memo1.Gutter.ShowLineNumbers:= true;   //---> reflection to box!
```

You simply put the line above on the boot script and make sure the ini file has it set to Yes. BOOTSCRIPT=Y //enabling load a boot script

“Wise men speak because they have something to say; Fools, because they have to say something”. -Plato

Feedback @ max@kleiner.com

Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

https://github.com/maxkleiner/maXbox3/releases

Here are 5 actions to make your computer more secure:

1. Use an antivirus program and keep it up to date
2. Do not open email messages from unknown sources or suspicious attachments even if you know the sender
3. Use a personal firewall
4. Keep your software up to date and enable Windows automatic updates
5. Use a pop-up blocker with your browser

1.11 Appendix Study with BigInt Direct

```
// TODO: Copy a file in a connected share path
//this is 333^4096:


function GetBigIntDirect: string;
  //Unit mybigint
var mbResult: TMyBigInt;
    i: integer;
begin
  mbResult:= TMyBigInt.Create(333);
  try
    // Faktoren im Zaehler aufmultiplizieren  ---> 2^12=4096
    for i:= 1  to 12 do begin
      mbResult.Multiply(mbresult, mbresult);
     //writeln(inttostr(i)+': '+mbresult.tostring);
     end;
   Result:= mbResult.ToString;
  finally
```

```
    //FreeAndNil(mbResult);
    mbResult.Free;
  end;
end;


 TMyBigInt = class
  private
    Len: Integer;
    Value: AnsiString;
    procedure Trim;
    procedure Shift(k: Integer);
    procedure MultiplyAtom(Multiplier1: TMyBigInt; Multiplier2: Integer);
  public
    constructor Create(iValue: Integer = 0);
    procedure Add(Addend1, Addend2: TMyBigInt);
    procedure Multiply(Multiplier1, Multiplier2: TMyBigInt); overload;
    procedure Multiply(Multiplier1: TMyBigInt; Multiplier2: Integer); overload;
    function ToString: string;
    procedure CopyFrom(mbCopy: TMyBigInt);
  end;
```