

## sbParser-SDK 3.62 for Win32

Creation and use of an sbParser object by HANDLE

### *csbparse.DLL* functions

char\* [GetParserVersionString](#)(char \*strbuffer);  
char\* [GetParserVersionNumber](#)(char \*strbuffer);

bool [InitSTDsbParserDLL](#)();  
bool [DeinitSTDsbParserDLL](#)();

### sbParser functions

HANDLE [CreateNewParser](#)(char \*FunctionIn, int VarCountIn,  
char \*VarDescIn);  
HANDLE [CreateNewParser](#)(char \*FunctionIn, int BaseIn = 10);  
HANDLE [CreateNewParser](#)(char \*FunctionIn, int BaseIn, int VarCountIn,  
char \*VarDescIn);

bool [DeleteParser](#)(HANDLE hParser);

bool [SetVarCont](#)(HANDLE hParser, const char \*AVarDescIn,  
long double VarContIn);

Annotation: Use dSetVarCont(...) if your compiler doesn't support long double (80bit)

bool [dSetVarCont](#)(HANDLE hParser, const char \*AVarDescIn,  
double VarContIn);

long double [GetResult](#)(HANDLE hParser);

long double [GetResult](#)(HANDLE hParser, long double VarContIn);

long double [GetResultExt](#)(HANDLE hParser, long double VarContIn[]);

Annotation: Use dGetResultxxx(...) if your compiler doesn't support long double (80bit)

double [dGetResult](#)(HANDLE hParser);

double [dGetResult](#)(HANDLE hParser, double VarContIn);

double [dGetResultExt](#)(HANDLE hParser, double VarContIn[]);

const char\* [GetOriginFunc](#)(HANDLE hParser, char \*strbuffer);

bool [GetIsOriginFunc](#)(HANDLE hParser, char \*FunctionIn);

int [GetOriginVarCount](#)(HANDLE hParser);

bool [GetIsError](#)(HANDLE hParser);

int [GetGlobalError](#)(HANDLE hParser);

char\* [GetUnknownFunction](#)(HANDLE hParser, char \*strbuffer);

bool [SetBaseTo](#)(HANDLE hParser, int BaseIn);

bool [SetAngularUnitTo](#)(HANDLE hParser, int AngularUnitIn);

bool [SetNewFunction](#)(HANDLE hParser, char \*FunctionIn);

### Useful functions

bool [ConvertToBase](#)(char \*strbuffer, const long double value,

```
int b_point, int Baseln);
```

Annotation: Use dConvertToBase(...) if your compiler doesn't support long double (80bit)

```
bool dConvertToBase(char *strbuffer, const double value,  
int b_point, int Baseln);
```

```
long double ConvertBaseTold(const char *strValue, int Baseln,  
char **endptr = NULL);
```

```
long double ConvertBaseTold(const char *strValue, int Baseln,  
int *errPos);
```

Annotation: Use ConvertBaseTod(...) if your compiler doesn't support long double (80bit)

```
double ConvertBaseTod(const char *strValue, int Baseln,  
char **endptr = NULL);
```

```
double ConvertBaseTod(const char *strValue, int Baseln,  
int *errPos);
```

```
unsigned int SetFPError(bool Enable = false);
```

```
char* ldtochar(char *strbuffer, const long double value,  
int point = 0);
```

Annotation: Use dtochar(...) if your compiler doesn't support long double (80bit)

```
char* dtochar(char *strbuffer, const double value,  
int point = 0);
```

```
long double chartold(const char *strValue, char **endptr = NULL);
```

```
long double chartold(const char *strValue, int *errPos);
```

Annotation: Use chartod(...) if your compiler doesn't support long double (80bit)

```
double chartod(const char *strValue, char **endptr = NULL);
```

```
double chartod(const char *strValue, int *errPos);
```



[SoftBase@T-Online.de](http://home.t-online.de/home/SoftBase/serv02_f.htm)

[http://home.t-online.de/home/SoftBase/serv02\\_f.htm](http://home.t-online.de/home/SoftBase/serv02_f.htm)

```
bool      InitSTDsbParserDLL();
bool      DeinitSTDsbParserDLL();
```

*NAME*

**InitSTDsbParserDLL / DeinitSTDsbParserDLL**

*DESCRIPTION* ([\*Example\*](#))

Loads or unloads the library *csbparse.DLL* and makes functions of the library available.

At your own program's beginning the function `InitSTDsbParserDLL` must be called. `DeinitSTDsbParserDLL` must be called at the end.

*RETURN VALUE*

<b>bool</b>	-	If the function succeeds, the return value is true. Otherwise it is false.
-------------	---	--

```
char*      GetParserVersionString(char *strbuffer);
char*      GetParserVersionNumber(char *strbuffer);
```

*NAME*

## **GetVersionString / GetVersionNumber**

*DESCRIPTION (Example: see C++ Builder example)*

Returns the version string or number of the *csbparse.DLL*.

*PARAMETERS*

<b>char*</b> strbuffer	-	Points to the buffer that will receive the string.
------------------------	---	--

*RETURN VALUE*

<b>char*</b>	-	The return value is the version string or number.
--------------	---	---

**class sbParser;**

NAME

## **sbParser ([Overview](#))**

DESCRIPTION

class sbParser (the creation and use of an sbParser object by HANDLE is explained here)

The class sbParser is an object for the calculation of functions of any length with a maximum of 24 variables.

In the function large and small letters are distinguished. Using subfunctions, constant factors and variables it must be considered, that e.g. sin(5) is recognized but SIN(5) is not. If only variable x is declared, X can not be interpreted.

SYNTAX

Possible subfunctions:

**sin, cos, tan, cot,**  
**asin, acos, atan, acot,**  
**sinh, cosh, tanh, coth,**  
**arsinh, arcosh, artanh, arcoth,**  
**sqr** (Square), **sqrt** (Square root),  
**exp, lg, ln,**  
**sgn** (Signum),  
**abs** (absolute value),  
**round** (Rounds E.g. round(5.8) = 6),  
**ceil** (Rounds up E.g. ceil(5.8) = 6),  
**floor** (Rounds down E.g. floor(5.8) = 5),  
**fac** (Factorial E.g. fac(5) = 120),  
**rand** (Random number in the field [0,c] E.g. rand(5) = 4),  
**GAMMA** (The Gamma-function E.g. GAMMA(5.7) ~ 72.5276)  
Annotation: The gamma function is implemented for values with one decimal place.

and  
**!** (Factorial E.g. 5! = 120),  
**^** (Power E.g. 3^2 = 9),  
Annotation:  $x^2$  equals  $x^2$  and  $x^3$  equals  $x^3$ )  
**%** (Percent E.g. 5+10% = 5.5 or 5\*10% = 0.5)

Possible constant factors are:

**pi, e.**

Characters within "" are ignored i.e. they are regarded as a comment.

E.g. in "**Costs 1996 are** 85.80\$" only 85.80 is considered.

Possible brackets are:

(,) and [,] where no priority exists.

Abriding spelling can be used in case of variables which only consist of one character e.g. 3.175x, x(..) , 5.635(...).

The missing operator is always interpreted as \*.

Additional subfunctions to built requests:

<b>ispos</b>	Value <= 0	Value > 0
<b>isposq</b>	Value < 0	Value >= 0
<b>isneg</b>	Value >= 0	Value < 0
<b>isnegq</b>	Value > 0	Value <= 0
<b>isnull</b>	Value != 0 (i.e. Value <> 0)	Value == 0
<b>not (==isnull)</b>	Value != 0	Value == 0

E.g.: ((Value1 < 3) AND (Value2 in [1;100]))  
is equal to  
(Value1-3 < 0) \* (Value2-1 >= 0) \* (Value2-100 <= 0),  
because \*==AND, + == OR.  
And this is finally equal to  
isneg(Value1-3) \* isposq(Value2-1) \* isnegq(Value2-100)

1	10	1 (== true)
3	1	0 (== false)
-10	-1	0 (== false)

## EXAMPLES

The examples are always clipped with [InitSTDsbParserDLL](#) and [DeinitSTDsbParserDLL](#). Each of these functions must only be called once in the program.

The examples uses the long double versions of the functions. You have to use the double versions if your compiler doesn't support long double (80bit). For further information see the document "readme.htm".

### 1.Example (simple calculation without variables)

```
//Link necessary header
#include <astsbpar.h>

//Loading of the csbparse.dll
InitSTDsbParserDLL();

//Creating an sbParser object and receipt its HANDLE
HANDLE hParser;
hParser = CreateNewParser("5+3^2+round(12.76)");

long double Result;
if (GetIsError(hParser))
{
    //an error occurred (However, it can not occur in the case of
    //the above function.)
    Result = 0;
}
else
{
    //Compute the result
    Result = GetResult(hParser);
}

//Conversion of the number to a string, e.g. for output
```

```
char strResult[30];  
ldtochar(strResult, Result, 10, true);
```

```
//Result is then 27  
//Delete of the sbParser object  
DeleteParser(hParser);
```

```
//Unloading of the csbparse.dll  
DeinitSTDsbParserDLL();
```

## 2.Example (variables which only consist of one character)

```
//Link necessary header  
#include <astsbpar.h>
```

```
//Loading of the csbparse.dll  
InitSTDsbParserDLL();
```

```
//Creating an sbParser object and receipt its HANDLE  
HANDLE hParser;  
hParser = CreateNewParser("-sin(x*cos(u))^(1/w)",3,"xuw");
```

```
//Declaring the variable contents  
long double VarIn[3];
```

```
VarIn[0] = 0.5; // it is the value of x  
VarIn[1] = 0.25; // it is the value of u  
VarIn[2] = -0.75; // it is the value of w
```

```
long double Result;  
if (GetIsError(hParser))  
{  
    //an error occurred (However, it can not occur in the case of  
    //the above function.)  
    Result = 0;  
}  
else  
{  
    //Compute the result  
    Result = GetResultExt(hParser,VarIn);  
}
```

```
//Conversion of the number to a string, e.g. for output  
char strResult[30];  
ldtochar(strResult, Result, 15, true);
```

```
//Result is then approx. -2.77007277434915  
//Delete of the sbParser object  
DeleteParser(hParser);
```

```
//Unloading of the csbparse.dll  
DeinitSTDsbParserDLL();
```

## 3.Example (variables which consist of more than one character)

```
//Link necessary header
```

```
#include <astsbpar.h>
```

```
//Loading of the csbparse.dll
```

```
InitSTDsbParserDLL();
```

```
//Creating an sbParser object and receipt its HANDLE
```

```
HANDLE hParser;
```

```
hParser = CreateNewParser("(Costs1994+Costs1995)-Costs1993*1.25",3,  
    "Costs1993 Costs1994 Costs1995");
```

```
//Declaring the variable contents
```

```
long double VarIn[3];
```

```
VarIn[0] = 1032.50;    // are the Costs1993
```

```
VarIn[1] = 2423.59;    // are the Costs1994
```

```
VarIn[2] = 285.3;      // are the Costs1995
```

```
long double Result;
```

```
if (GetIsError(hParser))
```

```
{
```

```
    //an error occurred (However, it can not occur in the case of  
    //the above function.)
```

```
    Result = 0;
```

```
}
```

```
else
```

```
{
```

```
    //Compute the result
```

```
    Result = GetResultExt(hParser,VarIn);
```

```
}
```

```
//Conversion of the number to a string, e.g. for output
```

```
char strResult[30];
```

```
ldtochar(strResult, Result, 8, true);
```

```
//Result is then 1418.265;
```

```
//Delete of the sbParser object
```

```
DeleteParser(hParser);
```

```
//Unloading of the csbparse.dll
```

```
DeinitSTDsbParserDLL();
```

**HANDLE**      **CreateNewParser(char \*FunctionIn, int VarCountIn,  
char \*VarDescIn);**

*NAME*

**CreateNewParser** (optional with variables)

*DESCRIPTION* ([Example](#))

Creates an [sbParser](#) object with the specified attributes and returns its HANDLE.

*PARAMETERS*

- |                         |   |   |
|-------------------------|---|---|
| <b>char *FunctionIn</b> | - | Points to a null-terminated string to be used as the <a href="#"><u>function</u></a> to work with.  |
| <b>int VarCountIn</b>   | - | Specifies the number of variables to work with (max.24). E.g. if the variables are "x_Value y_Value z_Value" or "xyz" VarCountIn must be set to 3.  |
| <b>char *VarDescIn</b>  | - | Points to a null-terminated string specifying the names of the working variables. Generally the variables must be separated by a blank. E.g. the variables x_Value, y_Value, z_Value must be set with "x_Value y_Value z_Value". If each of them only consists of one character the enumeration of the variables can be shortened by leaving the blank. E.g. the variables x, y, z can be set with "xyz". (Note: To use the short form you should make sure there's no variable consisting of more than one character!) |

*RETURN VALUE*

- |               |   |  |
|---------------|---|--|
| <b>HANDLE</b> | - | The return value is the HANDLE of the created sbParser object.<br><a href="#"><u>GetIsError()</u></a> should be called to check whether an error has occurred. To get more information about the error, call <a href="#"><u>GetGlobalError()</u></a> . |
|---------------|---|--|

**HANDLE**      **CreateNewParser**(char \*FunctionIn, int BaseIn, int VarCountIn,  
char \*VarDescIn);

NAME

**CreateNewParser** (based upon the number system with variables)

DESCRIPTION (Example: see C++ Builder example)

Creates an sbParser object with the specified attributes and returns its HANDLE.

PARAMETERS

<b>char</b> *FunctionIn	-	Points to a null-terminated string to be used as the <u>function</u> to work with.
<b>int</b> BaseIn	-	Specifies the number system the values given by the function string are based upon: 10 represents decimal, 2 represents binary, 16 represents hexadecimal, 8 represents octal.
<b>int</b> VarCountIn	-	Specifies the number of variables to work with (max.24). E.g. if the variables are "x_Value y_Value z_Value" or "xyz" VarCountIn must be set to 3.
<b>char</b> *VarDescIn	-	Points to a null-terminated string specifying the names of the working variables. Generally the variables must be separated by a blank. E.g. the variables x_Value, y_Value, z_Value must be set with "x_Value y_Value z_Value". If each of them only consists of one character the enumeration of the variables can be shortened by leaving the blank. E.g. the variables x, y, z can be set with "xyz". (Note: To use the short form you should make sure there's no variable consisting of more than one character!)

RETURN VALUE

<b>HANDLE</b>	-	The return value is the HANDLE of the created sbParser object. <u>GetIsError</u> () should be called to check whether an error has occurred. To get more information about the error, call <u>GetGlobalError</u> ().
---------------	---	---

**HANDLE**      **CreateNewParser**(char \*FunctionIn, int BaseIn = 10);

*NAME (Example: see Borland C++ example)*

**CreateNewParser** (based upon the number system)

*DESCRIPTION*

Creates an sbParser object with the specified attributes and returns its HANDLE.

*PARAMETERS*

<b>char</b> *FunctionIn	-	Points to a null-terminated string to be used as the <u>function</u> to work with.
<b>int</b> BaseIn	-	Specifies the number system the values given by the function string are based upon:
	10	represents decimal,
	2	represents binary,
	16	represents hexadecimal,
	8	represents octal.

*RETURN VALUE*

<b>HANDLE</b>	-	The return value is the HANDLE of the created sbParser object. <u>GetIsError()</u> should be called to check whether an error has occurred. To get more information about the error, call <u>GetGlobalError()</u> .
---------------	---	--

**bool**                    **DeleteParser**(**HANDLE** hParser);

*NAME*

## **DeleteParser**

*DESCRIPTION* ([\*Example\*](#))

Deletes an sbParser object.

*PARAMETERS*

**HANDLE** hParser   -        Identifies the sbParser object.

*RETURN VALUE*

**bool**                                -        If the function succeeds, the return value is true. If the given handle is not valid the return value is false.

**bool**                   **SetVarCont**(**HANDLE** hParser, **const char** \*AVarDescIn,  
                          **long double** VarContIn);

*NAME*

## **SetVarCont**

### *DESCRIPTION*

Sets the value for a variable of the sbParser object declared with [CreateNewParser\(...\)](#).  
(The calculation starts by calling the function [GetResult\(\)](#).)

### *PARAMETERS*

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>char</b> *AVarDescIn	-	Points to a null-terminated string which identifies the variable to be set.
<b>long double</b> VarContIn	-	Specifies the value for the variable.

### *RETURN VALUE*

<b>bool</b>	-	If the function succeeds, the return value is true. If the given handle is not valid or the variable does not exist the return value is false.
-------------	---	--

(Annotation: Use dSetVarCont(...) if your compiler doesn't support long double (80bit))

**bool**                   **dSetVarCont**(**HANDLE** hParser, **const char** \*AVarDescIn,  
                          **long double** VarContIn);

**long double     GetResult(HANDLE hParser);**

*NAME*

## **GetResult**

*DESCRIPTION* ([Example](#))

Solves the function with the variables declared by [CreateNewParser](#)(...) and with the values that were defined by calling [SetVarCont](#)(...).  
(If some variables are not defined by [SetVarCont](#)(...) the values of them are set automatically to 0.)

*PARAMETERS*

**HANDLE hParser**     -     Identifies the sbParser object.

*RETURN VALUE*

**long double**     -     If the function succeeds, the return value is the result of the calculation.  
[GetIsError](#)() should be called to check whether an error has occurred. To get more information about the error, call [GetGlobalError](#)().

(Annotation: Use dGetResult() if your compiler doesn't support long double (80bit))  
**double             dGetResult(HANDLE hParser);**

**long double     GetResult(HANDLE hParser, long double VarContIn);**

*NAME*

## **GetResult**

*DESCRIPTION*

Solves the function with the first variable declared by  
[CreateNewParser](#)(...) and with its value.  
(Any other variable is set automatically to 0.)

*PARAMETERS*

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>long double</b> VarContIn	-	Specifies value for the first variable.

*RETURN VALUE*

<b>long double</b>	-	If the function succeeds, the return value is the result of the calculation. <a href="#">GetIsError</a> () should be called to check whether an error has occurred. To get more information about the error, call <a href="#">GetGlobalError</a> ().
--------------------	---	---

(Annotation: Use dGetResult(...) if your compiler doesn't support long double (80bit)  
**double             dGetResult(HANDLE hParser, long double VarContIn);**)

**long double**     **GetResultExt**(**HANDLE** hParser, **long double** VarContIn[]);

NAME

## **GetResultExt**

DESCRIPTION ([Example](#))

Solves the function with the variables declared by  
[CreateNewParser](#)(...) and with the given values.

PARAMETERS

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>long double</b> VarContIn[]	-	Pointer to an array of long double defining the values of the variables.

RETURN VALUE

<b>long double</b>	-	If the function succeeds, the return value is the result of the calculation. <a href="#"><u>GetIsError</u></a> () should be called to check whether an error has occurred. To get more information about the error, call <a href="#"><u>GetGlobalError</u></a> ().
--------------------	---	--

(Annotation: Use dGetResultExt(...) if your compiler doesn't support long double (80bit)  
**double**             **dGetResultExt**(**HANDLE** hParser, **long double** VarContIn[]);)

```
const char*    GetOriginFunc(HANDLE hParser, char *strbuffer);
```

*NAME*

## **GetOriginFunc**

*DESCRIPTION*

Copies the string of the original function specified by  
[CreateNewParser\(...\)](#) into a buffer.

*PARAMETERS*

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>char</b> *strbuffer	-	Points to the buffer that will receive the string.

*RETURN VALUE*

<b>const char*</b>	-	The return value is the original function string.
--------------------	---	---

**bool**                   **GetIsOriginFunc**(**HANDLE** hParser, **char** \*FunctionIn);

*NAME*

## **GetIsOriginFunc**

*DESCRIPTION*

Determines whether the specified function string is the original function.

*PARAMETERS*

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>char</b> *FunctionIn	-	Points to a null-terminated string that will be tested.

*RETURN VALUE*

<b>bool</b>	-	If the string is the original function, the return value is true. Otherwise it is false.
-------------	---	--

**int      GetOriginVarCount (HANDLE hParser);**

*NAME*

## **GetOriginVarCount**

*DESCRIPTION*

Returns the original number of variables specified by  
[CreateNewParser\(...\)](#).

*PARAMETERS*

**HANDLE** hParser    -      Identifies the sbParser object.

*RETURN VALUE*

**bool**                                -      The return value is the original number of  
variables.

**bool**                    **GetIsError**(**HANDLE** hParser);

*NAME*

## **GetIsError**

*DESCRIPTION* ([Example](#))

Determines whether an error has occurred. Call [GetGlobalError](#)() to get extended error information.

*PARAMETERS*

**HANDLE** hParser    -            Identifies the sbParser object.

*RETURN VALUE*

**bool**                    -            If an error has occurred, the return value is true. Otherwise it is false.

**int      GetGlobalError(HANDLE hParser);**

NAME

## GetGlobalError

DESCRIPTION (Example: see Borland C++ or C++ Builder example)

The GetGlobalError function returns the sbParser object error code value. Possible errors are syntax- and calculation errors. Syntax errors [S] can be checked after creation of an sbParser Object. Calculation errors [C] can be checked after calling calculating functions [GetResult\(...\)](#) or [GetResultExt\(...\)](#).

PARAMETERS

**HANDLE hParser**    -      Identifies the sbParser object.

RETURN VALUE

<b>int</b>	-	The return value is the sbParser object error.
[S]	-1	The function string is a value. <u>This is not an error!!!</u> (...but sometimes useful) E.g. 3.654 is a value, 3+6 is not.
[S][C]	0	No error has occurred.
[S]	1	The function string contains an unknown subfunction. (This subfunction can be determined by calling <a href="#">GetUnknownFunction</a> ().)
[S]	2	Incorrect brackets are set in the function string.
[S]	3	The function string contains an unknown constant factor or variable.
[C]	4	A multiplication causes an error.
[C]	5	A division causes an error.
[C]	6	The result of the calculation is not valid.
[C]	7	An addition causes an error.
[C]	8	A subtraction causes an error.
[S]	9	An invalid comment in the function string. See above <a href="#">sbParser</a> syntax for "".
[C]	10	The specified derivation is out of range.
[S]	11	Work was aborted by user.
[S]	12	The specified number of variables is not the number of the declared variables. If the number of variables exceeds the maximum this error value is set, too.
[S]	13	A +, -, * or / is set incorrectly in the function string.
[S]	14	The factorial function (!) is set incorrectly in the function string.
[S]	15	An unexpected end in the function string.
[S]	16	A subfunction is set incorrectly in the function string.
[S]	18	The function string contains an unknown character.



**char\***            **GetUnknownFunction**(**HANDLE** hParser, **char** \*strbuffer);

*NAME*

## **GetUnknownFunction**

*DESCRIPTION*

If an unknown subfunction error (see above [GetGlobalError\(\)](#) == 1) has occurred [GetUnknownFunction](#)(...) copies the string of the subfunction into a buffer.

*PARAMETERS*

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>char</b> *strbuffer	-	Points to the buffer that will receive the string.

*RETURN VALUE*

<b>const char*</b>	-	The return value is the unknown subfunction string.
--------------------	---	---

**bool**                    **SetBaseTo**(**HANDLE** hParser, **int** BaseIn);

*NAME*

## **SetBaseTo**

*DESCRIPTION (Example: see C++ Builder example)*

Specifies the number system the values given by the function string are based upon.

However calculation results are returned as long double the function

[ConvertToBase](#)(...) can be used to convert a value to a base.

### *PARAMETERS*

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>int</b> BaseIn	-	Specifies the number system:
	10	represents decimal,
	2	represents binary,
	16	represents hexadecimal,
	8	represents octal.

### *RETURN VALUE*

<b>bool</b>	-	If the function succeeds, the return value is true. Otherwise it is false.
-------------	---	--

**bool**                   **SetAngularUnitTo**(**HANDLE** hParser, **int** AngularUnitIn);

*NAME*

## **SetAngularUnitTo**

*DESCRIPTION (Example: see C++ Builder example)*

Specifies the angular unit the trigonometric subfunctions are interpreted with.

*PARAMETERS*

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>int</b> AngularUnitIn	-	Specifies the angular unit:
	1	sets angular unit to degree,
	2	sets angular unit to radian,
	3	sets angular unit to centesimal degree.

*RETURN VALUE*

<b>bool</b>	-	If the function succeeds, the return value is true. Otherwise it is false.
-------------	---	--

**bool**                    **SetNewFunction**(**HANDLE** hParser, **char** \*FunctionIn);

*NAME*

## **SetNewFunction**

*DESCRIPTION (Example: see C++ Builder example)*

Specifies a new function for an existing sbParser object.

*PARAMETERS*

<b>HANDLE</b> hParser	-	Identifies the sbParser object.
<b>char</b> *FunctionIn	-	Points to a null-terminated string to be used as the function to work with.

*RETURN VALUE*

<b>bool</b>	-	If the function succeeds, the return value is true. Otherwise it is false. <u><a href="#">GetIsError()</a></u> should be called to check whether an error has occurred. To get more information about the error, call <u><a href="#">GetGlobalError()</a></u> .
-------------	---	--

```
bool ConvertToBase(char *strbuffer, const long double value,  
int b_point, int BaseIn);
```

## NAME

## ConvertToBase

DESCRIPTION (Example: see C++ Builder example)

Converts a long double value to a specified base and returns it as a string.

## PARAMETERS

<b>char *strbuffer</b>	-	Points to the buffer that will receive the string.
<b>const long double value</b>	-	Specifies the long double value.
<b>int b_point</b>	-	Specifies the maximum number of digits for the result.
<b>int Baseln</b>	-	Specifies the number system:
	10	represents decimal,
	2	represents binary,
	16	represents hexadecimal,
	8	represents octal.

### RETURN VALUE

- If the function succeeds, the return value is true. Otherwise it is false.

(Annotation: Use `dConvertToBase (...)` if your compiler doesn't support long double (80bit))

```
bool dConvertToBase(char *strbuffer, const double value,
                   int b_point, int BaseIn);
```

```

long double    ConvertBaseTold(const char *strValue, int BaseIn,
                                char **endptr = NULL);
    or
long double    ConvertBaseTold(const char *strValue, int BaseIn,
                                int *errPos);

```

NAME

## ConvertBaseTold

DESCRIPTION (Example: see C++ Builder example)

Converts a string representing a value of a specified base to a long double.

### PARAMETERS

<b>const char</b> *strValue	-	Points to a null-terminated string which identifies the value to be converted.
<b>int</b> BaseIn	-	Specifies the number system: 10 represents decimal, 2 represents binary, 16 represents hexadecimal, 8 represents octal.
<b>char</b> **endptr	-	If the scan of the string is successful endptr == <b>NULL</b> . Otherwise *endptr is set onto the character that stops the scan. (*endptr = &stop_char). The pointer endptr is useful to find errors.
or		
<b>int</b> *errPos	-	If the scan of the string is successful errPos == -1. Otherwise errPos is the position of the character that stops the scan.

### RETURN VALUE

<b>long double</b>	-	If the function succeeds, the return value is the result of the conversion.
--------------------	---	---

(Annotation: Use ConvertBaseTod(...) if your compiler doesn't support long double (80bit)

```

double         ConvertBaseTod(const char *strValue, int BaseIn,
                                char **endptr = NULL);
    or
double ConvertBaseTold(const char *strValue, int BaseIn,
                        int *errPos);)

```

**unsigned int** SetFPError(**bool** Enable = [false](#));

*NAME*

## **SetFPError**

*DESCRIPTION*

Activates or deactivates floating point coprocessor errors at operations like +, -, \*, /. I.e. if an error has occurred and SetFPError(false) has been called the program will not be terminated.

*PARAMETERS*

<b>bool</b> Enable	-	Specifies whether to enable or disable the coprocessor errors.
--------------------	---	--

*RETURN VALUE*

<b>unsigned int</b>	-	see documentation of <a href="#">_control87(...)</a> .
---------------------	---	--

**char\*** **ldtochar**(**char** \*strbuffer, **const long double** value,  
**int** point = 0);

NAME

## ldtochar

DESCRIPTION ([Example](#))

Converts a long double value into a string.

PARAMETERS

<b>char</b> *strbuffer	-	Points to the buffer that will receive the string.
<b>const long double</b> value	-	Specifies the long double value to be converted.
<b>int</b> point	-	Specifies the maximum number of digits for the result. If point is 0 the best number of digits will be used.

RETURN VALUE

<b>char*</b>	-	If the function succeeds, the return value is the result of the conversion.
--------------	---	---

(Annotation: Use dtochar(...) if your compiler doesn't support long double (80bit))

**char\*** **dtochar**(**char** \*strbuffer, **const double** value,  
**int** point = 0);

**long double**    **chartold(const char \*strValue, char \*\*endptr = NULL);**  
or  
**long double**    **chartold(const char \*strValue, int \*errPos);**

*NAME*  
**chartold**

*DESCRIPTION (Example: see Borland C++ example)*  
Converts a string representing a value to a long double.

*PARAMETERS*

<b>const char</b> <b>*strValue</b>	-	Points to a null-terminated string which identifies the value to be converted.
<b>char **endptr</b>	-	If the scan of the string is successful endptr == NULL. Otherwise *endptr is set onto the character that stops the scan. (*endptr = &stop_char). The pointer endptr is useful to find errors.
or		
<b>int *errPos</b>	-	If the scan of the string is successful errPos == -1. Otherwise errPos is the position of the character that stops the scan.

*RETURN VALUE*

<b>long double</b>	-	If the function succeeds, the return value is the result of the conversion.
--------------------	---	---

(Annotation: Use chartod(...) if your compiler doesn't support long double (80bit)

**double**    **chartod(const char \*strValue, char \*\*endptr = NULL);**  
or  
**double**    **chartod(const char \*strValue, int \*errPos);)**

