

# double128 Math Library (SDK)

## Quick Guide

### Computations & Graphics, Inc.

5290 Windflower Lane  
Highlands Ranch, CO 80130, USA  
Email: [info@cg-inc.com](mailto:info@cg-inc.com)  
Web: [www.cg-inc.com](http://www.cg-inc.com)

## Introduction

Computations & Graphics, Inc. (CGI) double128 Math Library (SDK) is 128-bit quad-precision math library that implements approximately 32 decimal place floating point arithmetic for Microsoft Visual Studio. The library includes a new floating point type double128 and a list of corresponding standard math functions such as sqrt(), pow(), sin() etc.. The implementation of the library is based on Intel C++ Compiler \_Quad data type. The following table shows a comparison among float, double and double128 types:

Parameter	single	double	double128
Format width in bits	32	64	128
Sign width in bits	1	1	1
Mantissa	24	53	113
Exponent width in bits	8	11	15
Max value	3.40282 x E38	1.79769 x E308	1.18973 x E4932
Min value	1.17549 x E-38	2.22507 x E-308	3.36210 x E-4932

Here is a list of the main features offered by the library:

- Quad-precision (128-bit) floating point arithmetic in Visual C++, C# and VB.NET.
- Supports basic I/O operations.
- Supports Visual Studio 2008, 2010, 2012, 2013 and 2015.
- Royalty-free on binary redistribution.

## C++ Interface

The C++ library contains both a 32-bit and a 64-bit Windows DLLs (double128.dll). The interface includes the following header files: double128\_api.h and double128.h. It also includes a double128Proj.lib for linking to your projects. The redistributable executable dll is double128Proj.dll.

The following lists the content of double128\_api.h:

```
#ifndef _DOUBLE128_API_H
#define _DOUBLE128_API_H

#ifdef DOUBLE128PRJ_EXPORTS
#define DOUBLE128PRJ_API __declspec(dllexport)
#else
#define DOUBLE128PRJ_API __declspec(dllimport)
#endif

#endif
```

The following lists the content of double128.h:

```
namespace cgiDouble128NS
{
    class DOUBLE128PRJ_API double128
    {
    public:
        enum {STRING_BUFFER_SIZE = 128};
        enum {QUAD_SIZE = 16};

        // constructors
        double128();
        double128(double val);
        double128(int val);
        double128(const double128& val);
        double128(const char* val);

        // destructor
        ~double128(void);

        // conversion operators
```

```

inline operator double()const;
inline operator int()const;
inline const char* getVal()const;
inline void copyMemoryFromQuad(const void* val);
inline void* copyMemoryToQuad(void*)const;
// you should supply string buffer size of at least STRING_BUFFER_SIZE = 128 chars here
inline char* str(char* buffer, int precision = 32, bool useScientific = false)const;

// operators
inline double128& operator +=(const double128& other);
inline double128& operator +=(double val);
inline double128& operator +=(int val);
inline double128& operator -=(const double128& other);
inline double128& operator -=(double val);
inline double128& operator -=(int val);
inline double128& operator *=(const double128& other);
inline double128& operator *=(double val);
inline double128& operator *=(int val);
inline double128& operator /=(const double128& other);
inline double128& operator /=(double val);
inline double128& operator /=(int val);
inline double128& operator ++(); // prefix
inline double128 operator ++(int); // postfix
inline double128& operator --();
inline double128 operator --(int);
private:
    char m_val[QUAD_SIZE];
};

// binary operators
DOUBLE128PRJ_API double128 operator -(const double128& val); // Negate -
DOUBLE128PRJ_API double128 operator +(const double128& val1, const double128& val2);
DOUBLE128PRJ_API double128 operator +(const double128& val1, double val2);
DOUBLE128PRJ_API double128 operator +(const double128& val1, int val2);
DOUBLE128PRJ_API double128 operator +(double val1, const double128& val2);
DOUBLE128PRJ_API double128 operator +(int val1, const double128& val2);
DOUBLE128PRJ_API double128 operator -(const double128& val1, const double128& val2);
DOUBLE128PRJ_API double128 operator -(const double128& val1, double val2);
DOUBLE128PRJ_API double128 operator -(const double128& val1, int val2);
DOUBLE128PRJ_API double128 operator -(double val1, const double128& val2);
DOUBLE128PRJ_API double128 operator -(int val1, const double128& val2);
DOUBLE128PRJ_API double128 operator *(const double128& val1, const double128& val2);

```

```

DOUBLE128PRJ_API double128 operator *(const double128& val1, double val2);
DOUBLE128PRJ_API double128 operator *(const double128& val1, int val2);
DOUBLE128PRJ_API double128 operator *(double val1, const double128& val2);
DOUBLE128PRJ_API double128 operator *(int val1, const double128& val12);
DOUBLE128PRJ_API double128 operator /(const double128& val1, const double128& val2);
DOUBLE128PRJ_API double128 operator /(const double128& val1, double val2);
DOUBLE128PRJ_API double128 operator /(const double128& val1, int val2);
DOUBLE128PRJ_API double128 operator /(double val1, const double128& val2);
DOUBLE128PRJ_API double128 operator /(int val1, const double128& val12);

```

```
// logical operators
```

```

DOUBLE128PRJ_API bool operator==(const double128& val1, const double128& val2);
DOUBLE128PRJ_API bool operator!=(const double128& val1, const double128& val2);
DOUBLE128PRJ_API bool operator> (const double128& val1, const double128& val2);
DOUBLE128PRJ_API bool operator>=(const double128& val1, const double128& val2);
DOUBLE128PRJ_API bool operator< (const double128& val1, const double128& val2);
DOUBLE128PRJ_API bool operator<=(const double128& val1, const double128& val2);
DOUBLE128PRJ_API bool operator==(const double128& val1, double val2);
DOUBLE128PRJ_API bool operator!=(const double128& val1, double val2);
DOUBLE128PRJ_API bool operator> (const double128& val1, double val2);
DOUBLE128PRJ_API bool operator>=(const double128& val1, double val2);
DOUBLE128PRJ_API bool operator< (const double128& val1, double val2);
DOUBLE128PRJ_API bool operator<=(const double128& val1, double val2);
DOUBLE128PRJ_API bool operator==(const double128& val1, int val2);
DOUBLE128PRJ_API bool operator!=(const double128& val1, int val2);
DOUBLE128PRJ_API bool operator> (const double128& val1, int val2);
DOUBLE128PRJ_API bool operator>=(const double128& val1, int val2);
DOUBLE128PRJ_API bool operator< (const double128& val1, int val2);
DOUBLE128PRJ_API bool operator<=(const double128& val1, int val2);
DOUBLE128PRJ_API bool operator==(double val1, const double128& val2);
DOUBLE128PRJ_API bool operator!=(double val1, const double128& val2);
DOUBLE128PRJ_API bool operator> (double val1, const double128& val2);
DOUBLE128PRJ_API bool operator>=(double val1, const double128& val2);
DOUBLE128PRJ_API bool operator< (double val1, const double128& val2);
DOUBLE128PRJ_API bool operator<=(double val1, const double128& val2);
DOUBLE128PRJ_API bool operator==(int val1, const double128& val2);
DOUBLE128PRJ_API bool operator!=(int val1, const double128& val2);
DOUBLE128PRJ_API bool operator> (int val1, const double128& val2);
DOUBLE128PRJ_API bool operator>=(int val1, const double128& val2);
DOUBLE128PRJ_API bool operator< (int val1, const double128& val2);
DOUBLE128PRJ_API bool operator<=(int val1, const double128& val2);

```

```
// math functions
```

```

DOUBLE128PRJ_API double128 fabs(const double128& val);
DOUBLE128PRJ_API double128 floor(const double128& val);
DOUBLE128PRJ_API double128 ceil(const double128& val);
DOUBLE128PRJ_API double128 sqrt(const double128& val);
DOUBLE128PRJ_API double128 trunc(const double128& val);
DOUBLE128PRJ_API double128 exp(const double128& val);
DOUBLE128PRJ_API double128 pow(const double128& val1, const double128& val2);
DOUBLE128PRJ_API double128 pow(const double128& val1, double val2);
DOUBLE128PRJ_API double128 pow(const double128& val1, int val2);
DOUBLE128PRJ_API double128 log(const double128& val);
DOUBLE128PRJ_API double128 log10(const double128& val);
DOUBLE128PRJ_API double128 sin(const double128& val);
DOUBLE128PRJ_API double128 cos(const double128& val);
DOUBLE128PRJ_API double128 tan(const double128& val);
DOUBLE128PRJ_API double128 asin(const double128& val);
DOUBLE128PRJ_API double128 acos(const double128& val);
DOUBLE128PRJ_API double128 atan(const double128& val);
DOUBLE128PRJ_API double128 sinh(const double128& val);
DOUBLE128PRJ_API double128 cosh(const double128& val);
DOUBLE128PRJ_API double128 tanh(const double128& val);
DOUBLE128PRJ_API double128 fmod(const double128& val1, const double128& val2);
DOUBLE128PRJ_API double128 fmod(const double128& val1, double val2);
DOUBLE128PRJ_API double128 fmod(const double128& val1, int val2);
DOUBLE128PRJ_API double128 atan2(const double128& val1, const double128& val2);
DOUBLE128PRJ_API double128 atan2(const double128& val1, double val2);
DOUBLE128PRJ_API double128 atan2(const double128& val1, int val2);

// arrays operations, a little faster
DOUBLE128PRJ_API double128 arrayAdd(const double128* pVal1, const double128* pVal2, int count);
DOUBLE128PRJ_API double128 arraySubtract(const double128* pVal1, const double128* pVal2, int count);
DOUBLE128PRJ_API double128 arrayMultiply(const double128* pVal1, const double128* pVal2, int count);
DOUBLE128PRJ_API double128 arrayDivide(const double128* pVal1, const double128* pVal2, int count);

// data range
DOUBLE128PRJ_API bool isnan(const double128& val);
DOUBLE128PRJ_API bool isinf(const double128& val);
DOUBLE128PRJ_API double128 quadMin();
DOUBLE128PRJ_API double128 quadMax();
DOUBLE128PRJ_API double128 quadLowest();
DOUBLE128PRJ_API double128 quadEpsilon();
}

```

## **double128 data type**

The type double128 is contained in the namespace cgiDouble128NS. It can be initialized from string, double or int data types. For maximum accuracy, double128 should be initialized from a string. double128 can be converted to double or int data types. It can also be output to a null-terminated string (see Basic I/O operations for double128 below).

double128 supports basic arithmetic operations (addition, subtraction, multiplication and division). Standard math functions such as sqrt(), pow(), sin() are available to operate on double128 data type.

arrayAdd() function sums the addition of corresponding items in two arrays. Similarly, arraySubtract(), arrayMultiply() and arrayDivide() sums the subtraction, multiplication and division of corresponding items in two arrays. These functions are provided for performance reasons.

isnan() function tests if a number is NAN (not a number)

isinf() function tests if a number is INF (infinite number)

quadMin() function returns 3.36210314311209350626267781732175260E-4932

quadMax() function returns 1.18973149535723176508575932662800702E4932

quadLowest() function returns -quadMax()

quadEpsilon() function returns 1.92592994438723585305597794258492732E-34

## **Basic I/O operations for double128**

```
double128::double128(const char* val)
```

The constructor above takes a null-terminated string.

```
char* double128::str(char* buffer, int precision = 32, bool useScientific = false) const
```

The member function above outputs double128 to a null-terminated string.

The arguments:

- buffer: user-supplied buffer that must be large enough to hold output value.
- precision: the precision after the decimal point.
- useScientific: output value in scientific notation.

Please note scientific notation will always be used in the following situations:

- If the value is positive and has an exponent greater than 30
- If the value is negative and has an exponent greater than 18
- If the value has an exponent less than -18

You should not call double128::str() function in the same statement unless different buffers are used. Therefore, it is generally a good idea to define an utility function in your application to wrap the double128::str() function. For example:

```
#include <string>
using namespace std;

inline string str(double128 val, int precision = 32, bool useScientific = false){
    char sz[128];
    val.str(sz, precision, useScientific);
    return string(sz);
}
```



## **Example 1 – calculate machine epsilon for double128**

The machine epsilon can be defined as the difference between 1 and the next larger representable number. For 32-bit float type, the machine epsilon is about 1.192092896e-07; For 64-bit float type, the machine epsilon is about 2.2204460492503131e-016; For 128-bit float type, the machine epsilon is about 1.9259299443872358530559779425849272e-34.

The following code snippet calculate the machine epsilon for double128 type.

```
#include <iostream>
using namespace std;

double128 machEps = double128("1.0");
while ((double128("1.0") + machEps) != double128("1.0")) {
    machEps /= double128("2.0");
}
machEps *= double128("2.0");
cout << "Calculated machine epsilon for double128 = " << str(machEps).c_str() << endl;
```

The above code outputs:

```
Calculated machine epsilon for double128 = 1.92592994438723585305597794258493E-0034
```



## **.NET Interface**

The .NET interface is a Class Library DLL (double128ProjCli.dll). It is written in C++/CLI and makes calls to the native Windows DLL (double128Proj.dll). The actual procedure to use this .NET interface is identical to C++ interface, except syntax differences.

Different versions of double128ProjCli.dll are provided to work with .NET Framework 3.5, 4.0, 4.5 and 4.6 on x86 and x64 CPUs. Make sure your project has reference to the correct double128ProjCli.dll. If your project is configured to build for both x86 and x64 CPUs, make sure to set “Copy Local” to false for the reference and manually copy the correct version of double128ProjCli.dll to the executable directory. Also make sure a copy of double128Proj.dll and double128ProjCli.dll is placed in the executable directory.

*For distribution, you may need to install Visual C++ x86 or x64 redistributables corresponding to Visual Studio as double128ProjCli.dll links dynamically with runtime library.*

## **Double128 data type**

The type Double128 is contained in the namespace double128ProjCli. It can be initialized from string, double or int data types. For maximum accuracy, Double128 should be initialized from a string. Double128 can be converted to double or int data types. It can also be output to a null-terminated string (see Basic I/O operations for Double128 below).

Double128 supports basic arithmetic operations (addition, subtraction, multiplication and division). Standard math functions such as sqrt(), pow(), sin() are available to operate on Double128 data type.

isnan() function tests if a number is NAN (not a number)

isinf() function tests if a number is INF (infinite number)

quadMin() function returns 3.36210314311209350626267781732175260E-4932

quadMax() function returns 1.18973149535723176508575932662800702E4932

quadLowest() function returns -quadMax()

quadEpsilon() function returns 1.92592994438723585305597794258492732E-34

## **Basic I/O operations for Double128**

```
Double128.Double128(string val)
```

The constructor above takes a string.

```
string Double128::ToString()  
string Double128::ToString(int precision, bool useScientific)
```

The member function above output Double128 to a string.

The arguments:

- precision: the precision after the decimal point (32 by default).
- useScientific: output value in scientific notation (false by default).

Please note scientific notation will always be used in the following situations:

- If the value is positive and has an exponent greater than 30
- If the value is negative and has an exponent greater than 18
- If the value has an exponent less than -18

## Example 1 – calculate machine epsilon for Double128

The machine epsilon can be informally defined as the difference between 1 and the next larger representable number. For 32-bit float type, the machine epsilon is about 1.192092896e-07; For 64-bit float type, the machine epsilon is about 2.2204460492503131e-016; For 128-bit float type, the machine epsilon is about 1.9259299443872358530559779425849272e-34.

The following code snippet calculate the machine epsilon for Double128 type.

```
using double128ProjCli;

Double128 machEps = new Double128("1.0");
while ((new Double128("1.0") + machEps) != new Double128("1.0"))
{
    machEps /= new Double128("2.0");
}
machEps *= new Double128("2.0");
Console.WriteLine("Calculated machine epsilon for Double128 = " + machEps.ToString());
```

The above code outputs:

```
Calculated machine epsilon for Double128 = 1.92592994438723585305597794258493E-0034
```

