**Using CString**

The topics in this section describe programming by using **CString**.

**CString**, **CStringA**, and **CStringW** are Microsoft-specific string classes that are available for general C++ development.

A **CStringA** object contains string data that is based on the **char** type, and supports single-byte and multibyte (**MBCS**) strings. Similarly, a **CStringW** object contains string data that is based on the **wchar\_t** type, and supports Unicode strings. A **CString** object supports either the **char** type or the **wchar\_t** type. Which one it supports depends on which one of the symbols, **MBCS** or **UNICODE**, is defined at compile time.

A **CString** object keeps character data in a **CStringData** object. **CString** accepts null-terminated C-style strings, but does not retain the null character in the stored character data. Instead, **CString** tracks string length. **CString** does provide a null terminator when it exports a C-style string. You can insert a null character amidst stored character data, but it may produce unexpected results.

**CString** is used in native projects. For managed-code (C++/CLI) projects, use **System::String**.

**CString**, **CStringA**, and **CStringW** are defined in **atlstr.h** and exported from **MFC90.dll**. **CStringT** is defined in **cstringt.h**. The following set of string classes can be used without linking an MFC library, with or without CRT support: **CAtlString**, **CAtlStringA**, and **CAtlStringW**.

The**CString**, **CStringA**, and **CStringW** classes are instances of a class template called **CStringT**. They take advantage of a template programming technique called specialization. Each is implemented as a typedef based on specializing the template class **CStringT** for the type of character data they support. They do not define their own methods. Instead, they rely on those defined by **CStringT**, specialized to work with the character data that they contain.

To add more capabilities than **CString**, **CStringA**, or **CStringW** currently offer, consider creating a subclass of **CStringT** that contains the additional features. Then create an instance that is similar to **CString** but is based on the subclass of **CStringT**.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**In This Section**](javascript:void(0))

[Basic CString Operations](http://msdn.microsoft.com/en-us/library/72b2swax.aspx)

Describes basic **CString** operations, including creating objects from C literal strings, accessing individual characters in a **CString**, concatenating two objects, and comparing **CString** objects.

[String Data Management](http://msdn.microsoft.com/en-us/library/8a994dfk.aspx)

Discusses using Unicode and MBCS with **CString**.

[CString Semantics](http://msdn.microsoft.com/en-us/library/sy280zek.aspx)

Explains how **CString** objects are used.

[CString Operations Relating to C-Style Strings](http://msdn.microsoft.com/en-us/library/awkwbzyc.aspx)

Describes manipulating the contents of a **CString** object like a C-style null-terminated string.

[Allocating and Releasing Memory for a BSTR](http://msdn.microsoft.com/en-us/library/xda6xzx7.aspx)

Discusses using memory for a **BSTR** and COM objects.

[CString Exception Cleanup](http://msdn.microsoft.com/en-us/library/ky89wkz5.aspx)

Explains that explicit cleanup in MFC 3.0 and later is no longer necessary.

[CString Argument Passing](http://msdn.microsoft.com/en-us/library/acttytz3.aspx)

Explains how to pass CString objects to functions and how to return **CString** objects from functions.

[Unicode and Multibyte Character Set (MBCS) Support](http://msdn.microsoft.com/en-us/library/ey142t48.aspx)

Discusses how MFC is enabled for Unicode and MBCS support.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Reference**](javascript:void(0))

[CStringT Class](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx) .

Provides reference information about the shared **CStringT** class.

[CSimpleStringT Class](http://msdn.microsoft.com/en-us/library/sddk80xf.aspx)

Provides reference information about the shared **CSimpleStringT** class.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Related Sections**](javascript:void(0))

[Strings (ATL/MFC)](http://msdn.microsoft.com/en-us/library/kda99ffc.aspx)

Contains links to topics that describe several ways to manage string data.

[Class Template Instantiation](http://msdn.microsoft.com/en-us/library/7y5ca42y.aspx)

**CString** is a typedef based on **CStringT**, an instance of a specialization of a class template.

[Strings (ATL/MFC)](http://msdn.microsoft.com/en-us/library/kda99ffc.aspx)

# Basic CString Operations

This topic explains the following basic [CString](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx) operations:

* [Creating CString objects from standard C literal strings](http://msdn.microsoft.com/en-us/library/72b2swax.aspx#_core_creating_cstring_objects_from_standard_c_literal_strings)
* [Accessing individual characters in a CString](http://msdn.microsoft.com/en-us/library/72b2swax.aspx#_core_accessing_individual_characters_in_a_cstring)
* [Concatenating two CString objects](http://msdn.microsoft.com/en-us/library/72b2swax.aspx#_core_concatenating_two_cstring_objects)
* [Comparing CString objects](http://msdn.microsoft.com/en-us/library/72b2swax.aspx#_core_comparing_cstring_objects)
* [Converting CString objects](http://msdn.microsoft.com/en-us/library/72b2swax.aspx#_core_converting_cstring_objects)

**Class CString** is based on class template [CStringT Class](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx). **CString** is a typedef of **CStringT**. More exactly, **CString** is a typedef of an *explicit specialization* of **CStringT**, which is a common way to use a class template to define a class. Similarly defined classes are **CStringA** and **CStringW**. For more information on explicit specialization, see [Class Template Instantiation](http://msdn.microsoft.com/en-us/library/7y5ca42y.aspx).

**CString**, **CStringA**, and **CStringW** are defined in atlstr.h. **CStringT** is defined in cstringt.h.

**CString**, **CStringA**, and **CStringW** each get a set of the methods and operators defined by **CStringT** for use with the string data they support. Some of the methods duplicate and, in some cases, surpass the string services of the C run-time libraries.

Note: **CString** is a native class. For a string class that is for use in a C++/CLI managed project, use **System.String**.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Creating CString Objects from Standard C Literal Strings**](javascript:void(0))

You can assign C-style literal strings to a **CString** just as you can assign one **CString** object to another.

* Assign the value of a C literal string to a **CString** object.

CString myString = \_T("This is a test");

* Assign the value of one **CString** to another **CString** object.

CString oldString = \_T("This is a test");

CString newString = oldString;

The contents of a **CString** object are copied when one **CString** object is assigned to another. Therefore, the two strings do not share a reference to the actual characters that make up the string. For more information about how to use **CString** objects as values, see [CString Semantics](http://msdn.microsoft.com/en-us/library/sy280zek.aspx).

|  |
| --- |
| **Note Note** |
| To write your application so that it can be compiled for Unicode or for ANSI, code literal strings by using the \_T macro. For more information, see [Unicode and Multibyte Character Set (MBCS) Support](http://msdn.microsoft.com/en-us/library/ey142t48.aspx). |

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Accessing Individual Characters in a CString**](javascript:void(0))

You can access individual characters in a **CString** object by using the **GetAt** and **SetAt** methods. You can also use the array element, or subscript, operator ( [ ] ) instead of**GetAt** to get individual characters. (This resembles accessing array elements by index, as in standard C-style strings.) **Index values for CString characters are zero-based.**

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Concatenating Two CString Objects**](javascript:void(0))

To concatenate two **CString** objects, use the concatenation operators (+ or +=), as follows.

CString s1 = \_T("This "); // Cascading concatenation

s1 += \_T("is a ");

CString s2 = \_T("test");

CString message = s1 + \_T("big ") + s2;

// Message contains "This is a big test".

At least one argument to the concatenation operators (+ or +=) must be a **CString** object, but you can use a constant character string (for example, "big") or a char (for example, 'x') for the other argument.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Comparing CString Objects**](javascript:void(0))

The **Compare** method and the == operator for **CString** are equivalent. **Compare**, **operator==**, and **CompareNoCase** are MBCS and Unicode aware; **CompareNoCase** is also case-insensitive. The **Collate** method of **CString** is locale-sensitive and is often slower than **Compare**. Use **Collate** only where you must abide by the sorting rules as specified by the current locale.

The following table shows the available [CString](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx) comparison functions and their equivalent Unicode/MBCS-portable functions in the C run-time library.

|  |  |  |
| --- | --- | --- |
| CString function | MBCS function | Unicode function |
| **Compare** | **\_mbscmp** | **wcscmp** |
| **CompareNoCase** | **\_mbsicmp** | **\_wcsicmp** |
| **Collate** | **\_mbscoll** | **wcscoll** |

The **CStringT** class template defines the relational operators (<, <=, >=, >, ==, and !=), which are available for use by **CString**. You can compare two **CStrings** by using these operators, as shown in the following example.

CString s1(\_T("Tom"));

CString s2(\_T("Jerry"));

ASSERT(s2 < s1);

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Converting CString Objects**](javascript:void(0))

For information about converting CString objects to other string types, see [How to: Convert Between Various String Types](http://msdn.microsoft.com/en-us/library/ms235631.aspx).

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Using CString with wcout**](javascript:void(0))

To use a CString with wcout you must explicitly cast the object to a const wchar\_t\* as shown in the following example:

CString cs("meow");

  wcout << (const wchar\_t\*) cs << endl;

**Without the cast, cs is treated as a void\* and wcout prints the address of the object**. This behavior is caused by subtle interactions between template argument deduction and overload resolution which are in themselves correct and conformant with the C++ standard.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**See Also**](javascript:void(0))

#### Reference

[CStringT Class](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx)

[Class Template Instantiation](http://msdn.microsoft.com/en-us/library/7y5ca42y.aspx)

[Explicit Specialization of Class Templates](http://msdn.microsoft.com/en-us/library/c401y1kb.aspx)

[Partial Specialization of Class Templates (C++)](http://msdn.microsoft.com/en-us/library/3967w96f.aspx)

[CStringT Class](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx)

#### Concepts

[How to: Convert Between Various String Types](http://msdn.microsoft.com/en-us/library/ms235631.aspx)

#### Other Resources

[Strings (ATL/MFC)](http://msdn.microsoft.com/en-us/library/kda99ffc.aspx)

# String Data Management

Visual C++ provides several ways to manage string data:

* [String Manipulation (CRT)](http://msdn.microsoft.com/en-us/library/f0151s4x.aspx) for working with C-style null-terminated strings
* Win32 API functions for managing strings
* MFC's class [CStringT Class](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx), which provides flexible, resizable string objects
* Class [CStringT Class](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx), which provides an MFC-independent string object with the same functionality as **CString**

Nearly all programs work with string data. MFC's **CString** class is often the best solution for flexible string handling. Starting with version 7.0, **CString** can be used in MFC or MFC-independent programs. Both the run-time library and **CString** support strings containing multibyte (wide) characters, as in Unicode or MBCS programming.

This article describes the general-purpose services that the class library provides related to string manipulation. Topics covered in this article include:

* [Unicode and MBCS Provide portability](http://msdn.microsoft.com/en-us/library/8a994dfk.aspx#_core_unicode_and_mbcs_provide_portability)
* [CStrings and const char Pointers](http://msdn.microsoft.com/en-us/library/8a994dfk.aspx#_core_cstrings_and_const_char_pointers)
* [CString Reference Counting](http://msdn.microsoft.com/en-us/library/8a994dfk.aspx#_core_cstring_reference_counting)

The [CStringT Class](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx) class provides support for manipulating strings. It is intended to replace and extend the functionality normally provided by the C run-time library string package. The **CString** class supplies member functions and operators for simplified string handling, similar to those found in Basic. The class also provides constructors and operators for constructing, assigning, and comparing **CStrings** and standard C++ string data types. Because **CString** is not derived from **CObject**, you can use **CString**objects independently of most of the Microsoft Foundation Class Library (MFC).

**CString** objects follow "**value semantics.**" A **CString** object represents a unique value. Think of a **CString** as an actual string, not as a pointer to a string.

A **CString** object represents a sequence of a variable number of characters. **CString** objects can be thought of as arrays of characters.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Unicode and MBCS Provide Portability**](javascript:void(0))

With MFC version 3.0 and later, MFC, including **CString**, is enabled for both Unicode and multibyte character sets (MBCS). This support makes it easier for you to write portable applications that you can build for either Unicode or ANSI characters. To enable this portability, **each character in a CString object is of type TCHAR**, which is defined as wchar\_t if you define the symbol **\_UNICODE** when you build your application, or as char if not. **A wchar\_t character is 16 bits wide**. MBCS is enabled if you build with the symbol **\_MBCS** defined. MFC itself is built with either the **\_MBCS** symbol (for the NAFX libraries) or the **\_UNICODE** symbol (for the UAFX libraries) defined.

|  |
| --- |
| **Note Note** |
| The **CString** examples in this and the accompanying articles on strings show literal strings properly formatted for Unicode portability, using the **\_T** macro, which translates the literal string to the form: |

L"literal string"

|  |
| --- |
| **Note Note** |
| which the compiler treats as a Unicode string. For example, the following code: |

CString strName = \_T("Name");

|  |
| --- |
| **Note Note** |
| is translated as a Unicode string if **\_UNICODE** is defined or as an ANSI string if not. For more information, see the article [Unicode and Multibyte Character Set (MBCS) Support](http://msdn.microsoft.com/en-us/library/ey142t48.aspx). |

**A CString object can store up to INT\_MAX (2,147,483,647) characters**. The **TCHAR** data type is used to get or set individual characters inside a **CString** object. Unlike character arrays, the **CString** class has a built-in memory allocation capability. This allows **CString** objects to automatically grow as needed (that is, you do not have to worry about growing a **CString** object to fit longer strings).

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**CStrings and const char Pointers**](javascript:void(0))

A **CString** object also can act like a literal C-style string (an **PCXSTR**, which is the same as **const char\*** if not under Unicode). The [CSimpleStringT::operator PCXSTR](http://msdn.microsoft.com/en-us/library/tk1z2hd9.aspx)conversion operator allows **CString** objects to be freely substituted for character pointers in function calls. The **CString( LPCWSTR** *pszSrc* **)** constructor allows character pointers to be substituted for **CString** objects.

No attempt is made to fold **CString** objects. If you make two **CString** objects containing Chicago, for example, the characters in Chicago are stored in two places. (This may not be true of future versions of MFC, so you should not depend on it.)

|  |
| --- |
| **Note Note** |
| Use the [CSimpleStringT::GetBuffer](http://msdn.microsoft.com/en-us/library/kt26tkzx.aspx) and [CSimpleStringT::ReleaseBuffer](http://msdn.microsoft.com/en-us/library/32x198c7.aspx) member functions when you need to directly access a **CString** as a nonconstant pointer to a character. |
| **Note Note** |
| Use the [CStringT::AllocSysString](http://msdn.microsoft.com/en-us/library/za1865s1.aspx) and [CStringT::SetSysString](http://msdn.microsoft.com/en-us/library/860zx2dt.aspx) member functions to allocate and set **BSTR** objects used in Automation (formerly known as OLE Automation). |

|  |
| --- |
| **Note Note** |
| Where possible, allocate **CString** objects on the frame rather than on the heap. This saves memory and simplifies parameter passing. |

The **CString** class is not implemented as a Microsoft Foundation Class Library collection class, though **CString** objects can certainly be stored as elements in collections.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**CString Reference Counting**](javascript:void(0))

As of MFC version 4.0, when [CStringT Class](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx) objects are copied, MFC increments a reference count rather than copying the data. This makes passing parameters by value and returning **CString** objects by value more efficient. These operations cause the copy constructor to be called, sometimes more than once. Incrementing a reference count reduces that overhead for these common operations and makes using **CString** a more attractive option.

As each copy is destroyed, the reference count in the original object is decremented. The original **CString** object is not destroyed until its reference count is reduced to zero.

You can use the **CString** member functions [CSimpleStringT::LockBuffer](http://msdn.microsoft.com/en-us/library/692ft902.aspx) and [CSimpleStringT::UnlockBuffer](http://msdn.microsoft.com/en-us/library/sk4as8ek.aspx) to disable or enable reference counting.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**See Also**](javascript:void(0))

#### Concepts

[General MFC Topics](http://msdn.microsoft.com/en-us/library/583ya1kc.aspx)

# CString Semantics

Even though [CString](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx) objects are dynamic objects that can grow, they act like built-in primitive types and simple classes. Each **CString** object represents a unique value.**CString** objects should be thought of as the actual strings rather than as pointers to strings.

You can assign one **CString**object to another. However, when you modify one of the two **CString** objects, the other **CString** object is not modified, as shown by the following example:

CString s1, s2;

s1 = s2 = \_T("hi there");

ASSERT(s1 == s2); // they are equal

s1.MakeUpper(); // Does not modify s2

ASSERT(s2[0] == \_T('h')); // s2 is still "hi there"

Note in the example that the two **CString** objects are considered "equal" because they represent the same character string. The **CString** class overloads the equality operator (==) to compare two **CString** objects based on their value (contents) rather than their identity (address).

# CString Operations Relating to C-Style Strings

A [CString](http://msdn.microsoft.com/en-us/library/ms174288.aspx) object contains character string data. **CString** inherits the set of the [methods and operators](http://msdn.microsoft.com/en-us/library/abzc9989.aspx) that are defined in the class template [CStringT](http://msdn.microsoft.com/en-us/library/5bzxfsea.aspx) to work with string data. (**CString** is a typedef that specializes **CStringT** to work with the kind of character data that **CString** supports.)

**CString** does not store character data internally as a C-style null-terminated string. Instead, **CString** tracks the length of character data so that it can more securely watch the data and the space it requires.

**CString** does accept C-style strings, and provides ways to access character data as a C-style string. This topic contains the following sections that explain how to use a**CString** object as if it were a C-style null-terminated string.

* [Converting to C-style null-terminated strings](http://msdn.microsoft.com/en-us/library/awkwbzyc.aspx#_core_using_cstring_as_a_c.2d.style_null.2d.terminated_string)
* [Working with standard run-time library string functions](http://msdn.microsoft.com/en-us/library/awkwbzyc.aspx#_core_working_with_standard_run.2d.time_library_string_functions)
* [Modifying CString contents directly](http://msdn.microsoft.com/en-us/library/awkwbzyc.aspx#_core_modifying_cstring_contents_directly)
* [Using CString objects with variable argument functions](http://msdn.microsoft.com/en-us/library/awkwbzyc.aspx#_core_using_cstring_objects_with_variable_argument_functions)
* [Specifying CString formal parameters](http://msdn.microsoft.com/en-us/library/awkwbzyc.aspx#_core_specifying_cstring_formal_parameters)

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Using CString as a C-Style Null-Terminated String**](javascript:void(0))

To use a **CString** object as a C-style string, cast the object to **LPCTSTR**. In the following example, the **CString** returns a pointer to a read-only C-style null-terminated string. The **strcpy** function puts a copy of the C-style string in the variable myString.

CString aCString = "A string";

char myString[256];

strcpy(myString, (LPCTSTR)aString);

You can use **CString** methods, for example, **SetAt**, to modify individual characters in the string object. However, the **LPCTSTR** pointer is temporary and becomes invalid when any change is made to **CString**. The **CString** can also go out of scope and be automatically deleted. **We recommend that you get a fresh LPCTSTR pointer of aCString object every time that you use one.**

Sometimes you may require a copy of **CString** data to modify directly. Use the more secured function **strcpy\_s** (or the Unicode/MBCS-portable **\_tcscpy\_s**) to copy the**CString** object into a separate buffer. This is where characters can be safely modified, as shown by the following example.

CString theString(\_T("This is a test"));

int sizeOfString = (theString.GetLength() + 1);

LPTSTR lpsz = new TCHAR[sizeOfString];

\_tcscpy\_s(lpsz, sizeOfString, theString);

//... modify lpsz as much as you want

|  |
| --- |
| **Note Note** |
| The third argument to strcpy\_s (or the Unicode/MBCS-portable \_tcscpy\_s) is either a constwchar\_t\* (Unicode) or a constchar\* (ANSI). The example above passes aCString for this argument. The C++ compiler automatically applies the conversion function defined for the CString class that converts a CString to an LPCTSTR. The ability to define casting operations from one type to another is one of the most useful features of C++. |

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Working with Standard Run-Time Library String Functions**](javascript:void(0))

You should be able to find a **CString** method to perform any string operation for which you might consider using the standard C run-time library string functions such as**strcmp** (or the Unicode/MBCS-portable **\_tcscmp**).

If you must use the C run-time string functions, you can use the techniques described in [Converting to C-Style Null-Terminated Strings](http://msdn.microsoft.com/en-us/library/awkwbzyc.aspx)\_core\_using\_cstring\_as\_a\_c.2d.style\_null.2d.terminated\_string. You can copy the **CString** object to an equivalent C-style string buffer, perform your operations on the buffer, and then assign the resulting C-style string back to a **CString** object.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Modifying CString Contents Directly**](javascript:void(0))

In most situations, you should use **CString** member functions to modify the contents of a **CString** object or to convert the **CString** to a C-style character string.

There are some situations where it makes sense to directly modify the **CString** contents, for example, when you work with operating-system functions that require a character buffer.

The **GetBuffer** and **ReleaseBuffer** methods offer access to the internal character buffer of a **CString** object and let you modify it directly. The following steps show how to use these functions for this purpose.

### To use GetBuffer and ReleaseBuffer to access the internal character buffer of a CString object

1. Call **GetBuffer** for a **CString** object and specify the length of the buffer you require.
2. Use the pointer returned by **GetBuffer** to write characters directly into the **CString** object.
3. Call **ReleaseBuffer** for the **CString** object to update all the internal **CString** state information, for example, the length of the string. After you modify the contents of a **CString** object directly, you must call **ReleaseBuffer** before you call any other **CString** member functions.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Using CString Objects with Variable Argument Functions**](javascript:void(0))

Some C functions take a variable number of arguments. A notable example is **printf\_s**. Because of the way this kind of function is declared, the compiler cannot be sure of the type of the arguments and cannot determine which conversion operation to perform on each argument. Therefore, it is essential that you use an explicit type cast when passing a **CString** object to a function that takes a variable number of arguments.

To use a **CString** object in a variable argument function, explicitly cast the **CString** to an **LPCTSTR** string, as shown in the following example.

CString kindOfFruit = \_T("bananas");

int howmany = 25;

\_tprintf\_s(\_T("You have %d %s\n"), howmany, (LPCTSTR)kindOfFruit);

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Specifying CString Formal Parameters**](javascript:void(0))

For most functions that need a string argument, it is best to specify the formal parameter in the function prototype as a const pointer to a character (**LPCTSTR**) instead of a**CString**. When a formal parameter is specified as a const pointer to a character, you can pass either a pointer to a **TCHAR** array, a literal string ["hi there"], or a **CString**object. The **CString** object will be automatically converted to an **LPCTSTR**. Any place you can use an **LPCTSTR**, you can also use a **CString** object.

You can also specify a formal parameter as a constant string reference (that is, constCString&) if the argument will not be modified. Drop the const modifier if the string will be modified by the function. If a default null value is desired, initialize it to the null string [""], as shown below:

void AddCustomer(const CString& name, const CString& address,

const CString& comment = \_T(""));

For most function results, you can simply return a **CString** object by value.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**See Also**](javascript:void(0))

#### Concepts

[CString Argument Passing](http://msdn.microsoft.com/en-us/library/acttytz3.aspx)

#### Other Resources

[Strings (ATL/MFC)](http://msdn.microsoft.com/en-us/library/kda99ffc.aspx)