

# Contents

[.NET programming with C++/CLI](#)

[Walkthrough: Compile a C++/CLI program that targets the CLR](#)

[C++/CLI tasks](#)

[C++/CLI tasks](#)

[How to: Create CLR empty projects](#)

[How to: Create CLR console applications](#)

[How to: Use tracking references in C++/CLI](#)

[How to: Use arrays in C++/CLI](#)

[How to: Define and consume classes and structs](#)

[C++ stack semantics for reference types](#)

[User-defined operators](#)

[User-defined conversions](#)

[initonly](#)

[How to: Define and use delegates](#)

[How to: Define and consume enums in C++/CLI](#)

[How to: Use events in C++/CLI](#)

[How to: Define an interface static constructor](#)

[How to: Declare override specifiers in native compilations](#)

[How to: Use properties in C++/CLI](#)

[How to: Use safe\\_cast in C++/CLI](#)

[Regular expressions](#)

[File handling and I/O](#)

[Graphics operations](#)

[Windows operations](#)

[Data access using ADO.NET](#)

[Native and .NET interoperability](#)

[Native and .NET interoperability](#)

[Interoperability with other .NET languages](#)

[Mixed \(native and managed\) assemblies](#)

[Mixed \(native and managed\) assemblies](#)

[How to: Migrate to -clr](#)

[How to: Compile MFC and ATL code by using -clr](#)

[Initialization of mixed assemblies](#)

[Library support for mixed assemblies](#)

[Performance considerations for interop \(C++\)](#)

[Application domains and Visual C++](#)

[Double thunking \(C++\)](#)

[Avoiding exceptions on CLR shutdown when consuming COM objects built with -clr](#)

[How to: Create a partially trusted application by removing dependency on the CRT library DLL](#)

[Using a Windows Form user control in MFC](#)

[Using a Windows Form user control in MFC](#)

[Windows Forms-MFC programming differences](#)

[Hosting a Windows Form user control in an MFC dialog box](#)

[Hosting a Windows Form user control in an MFC dialog box](#)

[How to: Create the user control and host in a dialog box](#)

[How to: Do DDX-DDV Data Binding with Windows Forms](#)

[How to: Sink Windows Forms events from native C++ classes](#)

[Hosting a Windows Forms User Control as an MFC View](#)

[Hosting a Windows Forms User Control as an MFC View](#)

[How to: Create the user control and host MDI View](#)

[How to: Add command routing to the Windows Forms control](#)

[How to: Call properties and methods of the Windows Forms control](#)

[Hosting a Windows Form user control as an MFC dialog box](#)

[Calling native functions from managed code](#)

[Calling native functions from managed code](#)

[Using explicit PInvoke in C++ \(DllImport attribute\)](#)

[Using explicit PInvoke in C++ \(DllImport attribute\)](#)

[How to: Call native DLLs from managed code using PInvoke](#)

[How to: Marshal strings using PInvoke](#)

[How to: Marshal structures using PInvoke](#)

[How to: Marshal arrays using PInvoke](#)

- [How to: Marshal function pointers using PInvoke](#)
- [How to: Marshal embedded pointers using PInvoke](#)
- [Using C++ interop \(implicit PInvoke\)](#)
  - [Using C++ interop \(implicit PInvoke\)](#)
  - [How to: Marshal ANSI strings using C++ interop](#)
  - [How to: Marshal Unicode strings using C++ interop](#)
  - [How to: Marshal COM strings using C++ interop](#)
  - [How to: Marshal structures using C++ interop](#)
  - [How to: Marshal arrays using C++ interop](#)
  - [How to: Marshal callbacks and delegates by using C++ interop](#)
  - [How to: Marshal embedded pointers using C++ interop](#)
  - [How to: Extend the marshaling library](#)
  - [How to: Access characters in a System::String](#)
  - [How to: Convert char \\* string to System::Byte array](#)
  - [How to: Convert System::String to wchar\\_t\\* or char\\*](#)
  - [How to: Convert System::String to standard string](#)
  - [How to: Convert standard string to System::String](#)
  - [How to: Obtain a pointer to byte array](#)
  - [How to: Load unmanaged resources into a byte array](#)
  - [How to: Modify reference class in a native function](#)
  - [How to: Determine if an image is native or CLR](#)
  - [How to: Add native DLL to global assembly cache](#)
  - [How to: Hold reference to value type in native type](#)
  - [How to: Hold object reference in unmanaged memory](#)
  - [How to: Detect -clr compilation](#)
  - [How to: Convert between System::Guid and \\_GUID](#)
  - [How to: Specify an out parameter](#)
  - [How to: Use a native type in a -clr compilation](#)
  - [How to: Declare handles in native types](#)
  - [How to: Wrap native class for use by C#](#)
- [Pure and verifiable code](#)
  - [Pure and verifiable code](#)

How to: Create verifiable C++ projects  
Using verifiable assemblies with SQL Server  
Converting projects from mixed mode to pure intermediate language

Serialization

Friend assemblies (C++)

Managed types

Reflection

Strong name assemblies (assembly signing)

Debug class

STL/CLR library reference

    STL/CLR library reference

    cliext namespace

    STL/CLR containers

Requirements for STL/CLR container elements

How to: Convert from a .NET collection to a STL/CLR container

How to: Convert from a STL/CLR container to a .NET collection

How to: Expose an STL/CLR container from an assembly

for each, in

    adapter (STL/CLR)

    algorithm (STL/CLR)

    deque (STL/CLR)

    functional (STL/CLR)

    hash\_map (STL/CLR)

    hash\_multimap (STL/CLR)

    hash\_multiset (STL/CLR)

    hash\_set (STL/CLR)

    list (STL/CLR)

    map (STL/CLR)

    multimap (STL/CLR)

    multiset (STL/CLR)

    numeric (STL/CLR)

    priority\_queue (STL/CLR)

[queue \(STL/CLR\)](#)

[set \(STL/CLR\)](#)

[stack \(STL/CLR\)](#)

[utility \(STL/CLR\)](#)

[vector \(STL/CLR\)](#)

[C++ support library](#)

[C++ support library](#)

[Overview of marshaling in C++](#)

[Overview of marshaling in C++](#)

[marshal\\_as](#)

[marshal\\_context class](#)

[msclr namespace](#)

[Resource management classes](#)

[Resource management classes](#)

[auto\\_gcroot](#)

[auto\\_gcroot](#)

[auto\\_gcroot class](#)

[swap function \(auto\\_gcroot\)](#)

[auto\\_handle](#)

[auto\\_handle](#)

[auto\\_handle class](#)

[swap function \(auto\\_handle\)](#)

[Synchronization \(lock class\)](#)

[Synchronization \(lock class\)](#)

[lock](#)

[lock](#)

[lock class](#)

[lock\\_when Enum](#)

[Calling functions in a specific application domain](#)

[Calling functions in a specific application domain](#)

[call\\_in\\_appdomain Function](#)

[com::ptr](#)

[com::ptr](#)

[com::ptr class](#)

[Exceptions in C++/CLI](#)

[Exceptions in C++/CLI](#)

[Basic concepts in using managed exceptions](#)

[Differences in exception handling behavior under -clr  
finally](#)

[How to: Catch exceptions in native code thrown from MSIL](#)

[How to: Define and install a global exception handler](#)

[Boxing](#)

[Boxing](#)

[How to: Explicitly request boxing](#)

[How to: Use gcnew to create value types and use implicit boxing](#)

[How to: Unbox](#)

[Standard conversions and implicit boxing](#)

# .NET programming with C++/CLI

9/20/2022 • 2 minutes to read • [Edit Online](#)

By default, CLR projects created with Visual Studio 2015 target .NET Framework 4.5.2. You can target .NET Framework 4.6 when you create a new project. In the **New Project** dialog, change the target framework in the dropdown at the top middle of the dialog. To change the target framework for an existing project, close the project, edit the project file (`.vcxproj`), and change the value of the Target Framework Version to 4.6. The changes take effect the next time you open the project.

In Visual Studio 2017, the default target .NET Framework is 4.6.1. The Framework version selector is at the bottom of the **New Project** dialog.

## Install C++/CLI support in Visual Studio 2017

C++/CLI itself isn't installed by default when you install a Visual Studio C++ workload. To install the component after Visual Studio is installed, open the Visual Studio Installer by selecting the Windows **Start** menu and searching for **visual studio installer**. Choose the **Modify** button next to your installed version of Visual Studio. Select the **Individual components** tab. Scroll down to the **Compilers, build tools, and runtimes** section, and select **C++/CLI support**. Select **Modify** to download the necessary files and update Visual Studio.

In Visual Studio 2019, the default target framework for .NET Core projects is 5.0. For .NET Frameworks projects, the default is 4.7.2. The .NET Framework version selector is on the **Configure your new project** page of the **Create a new project** dialog.

## Install C++/CLI support in Visual Studio 2019

C++/CLI itself isn't installed by default when you install a Visual Studio C++ workload. To install the component after Visual Studio is installed, open the Visual Studio Installer by selecting the Windows **Start** menu and searching for **visual studio installer**. Choose the **Modify** button next to your installed version of Visual Studio. Select the **Individual components** tab. Scroll down to the **Compilers, build tools, and runtimes** section, and select **C++/CLI support for v142 build tools (Latest)**. Select **Modify** to download the necessary files and update Visual Studio.

In Visual Studio 2022, the default target framework for .NET Core projects is 6.0. For .NET Frameworks projects, the default is 4.7.2. The .NET Framework version selector is on the **Configure your new project** page of the **Create a new project** dialog.

## Install C++/CLI support in Visual Studio 2022

C++/CLI itself isn't installed by default when you install a Visual Studio C++ workload. To install the component after Visual Studio is installed, open the Visual Studio Installer by selecting the Windows **Start** menu and searching for **visual studio installer**. Choose the **Modify** button next to your installed version of Visual Studio. Select the **Individual components** tab. Scroll down to the **Compilers, build tools, and runtimes** section, and select **C++/CLI support for v143 build tools (Latest)**. Select **Modify** to download the necessary files and update Visual Studio.

## In this section

[C++/CLI tasks](#)

[Native and .NET interoperability](#)

[Pure and verifiable code \(C++/CLI\)](#)

[Regular expressions \(C++/CLI\)](#)

[File handling and I/O \(C++/CLI\)](#)

[Graphics operations \(C++/CLI\)](#)

[Windows operations \(C++/CLI\)](#)

[Data access using ADO.NET \(C++/CLI\)](#)

[Interoperability with other .NET languages \(C++/CLI\)](#)

[Serialization \(C++/CLI\)](#)

[Managed types \(C++/CLI\)](#)

[Reflection \(C++/CLI\)](#)

[Strong Name assemblies \(assembly signing\) \(C++/CLI\)](#)

[Debug class \(C++/CLI\)](#)

[STL/CLR library reference](#)

[C++ support library](#)

[Exceptions in C++/CLI](#)

[Boxing \(C++/CLI\)](#)

## See also

[Native and .NET interoperability](#)

# Walkthrough: Compile a C++/CLI program that targets the CLR in Visual Studio

9/20/2022 • 3 minutes to read • [Edit Online](#)

By using C++/CLI you can create C++ programs that use .NET classes as well as native C++ types. C++/CLI is intended for use in console applications and in DLLs that wrap native C++ code and make it accessible from .NET programs. To create a Windows user interface based on .NET, use C# or Visual Basic.

For this procedure, you can type your own C++ program or use one of the sample programs. The sample program that we use in this procedure creates a text file named `textfield.txt`, and saves it to the project directory.

## Prerequisites

- An understanding of the fundamentals of the C++ language.
- In Visual Studio 2017 and later, C++/CLI support is an optional component. To install it, open the **Visual Studio Installer** from the Windows Start menu. Make sure that the **Desktop development with C++** tile is checked, and in the **Optional** components section, also check **C++/CLI Support**.

## Create a new project

The following steps vary depending on which version of Visual Studio you are using. To see the documentation for your preferred version of Visual Studio, use the **Version** selector control. It's found at the top of the table of contents on this page.

### To create a C++/CLI project in Visual Studio

1. In **Solution Explorer**, right-click on the top to open the **Create a New Project** dialog box.
2. At the top of the dialog, type **CLR** in the search box and then choose **CLR Empty Project** from the results list.
3. Choose the **Create** button to create the project.

### To create a C++/CLI project in Visual Studio 2017

1. Create a new project. On the **File** menu, point to **New**, and then click **Project**.
2. From the Visual C++ project types, click **CLR**, and then click **CLR Empty Project**.
3. Type a project name. By default, the solution that contains the project has the same name as the new project, but you can enter a different name. You can enter a different location for the project if you want.
4. Click **OK** to create the new project.

### To create a C++/CLI project in Visual Studio 2015

1. Create a new project. On the **File** menu, point to **New**, and then click **Project**.
2. From the Visual C++ project types, click **CLR**, and then click **CLR Empty Project**.
3. Type a project name. By default, the solution that contains the project has the same name as the new project, but you can enter a different name. You can enter a different location for the project if you want.
4. Click **OK** to create the new project.

## Add a source file

1. If **Solution Explorer** isn't visible, click **Solution Explorer** on the **View** menu.
2. Add a new source file to the project:
  - Right-click the **Source Files** folder in **Solution Explorer**, point to **Add**, and click **New Item**.
  - Click **C++ File (.cpp)** and type a file name and then click **Add**.
3. Click in the newly created tab in Visual Studio and type a valid Visual C++ program, or copy and paste one of the sample programs.

For example, you can use the [How to: Write a Text File \(C++/CLI\)](#) sample program (in the **File Handling and I/O** node of the Programming Guide).

If you use the sample program, notice that you use the `gcnew` keyword instead of `new` when creating a .NET object, and that `gcnew` returns a handle (`^`) rather than a pointer (`*`):

```
StreamWriter^ sw = gcnew StreamWriter(fileName);
```

For more information on C++/CLI syntax, see [Component Extensions for Runtime Platforms](#).

4. On the **Build** menu, click **Build Solution**.

The **Output** window displays information about the compilation progress, such as the location of the build log and a message that indicates the build status.

If you make changes and run the program without doing a build, a dialog box might indicate that the project is out of date. Select the checkbox on this dialog before you click **OK** if you want Visual Studio to always use the current versions of files instead of prompting you each time it builds the application.

5. On the **Debug** menu, click **Start without Debugging**.
6. If you used the sample program, when you run the program a command window is displayed that indicates the text file has been created.

The `textfield.txt` text file is now located in your project directory. You can open this file by using Notepad.

### NOTE

Choosing the empty CLR project template automatically set the `/clr` compiler option. To verify this, right-click the project in **Solution Explorer** and clicking **Properties**, and then check the **Common Language Runtime support** option in the **General** node of **Configuration Properties**.

## See also

- [C++ Language Reference](#)  
[Projects and build systems](#)

# C++/CLI Tasks

9/20/2022 • 2 minutes to read • [Edit Online](#)

The articles in this section of the documentation show how to use various features of C++/CLI.

## In This Section

- [How to: Create CLR Empty Projects](#)
- [How to: Create CLR Console Applications \(C++/CLI\)](#)
- [How to: Use Tracking References in C++/CLI](#)
- [How to: Use Arrays in C++/CLI](#)
- [How to: Define and Consume Classes and Structs \(C++/CLI\)](#)
- [C++ Stack Semantics for Reference Types](#)
- [User-Defined Operators \(C++/CLI\)](#)
- [User-Defined Conversions \(C++/CLI\)](#)
- [initonly \(C++/CLI\)](#)
- [How to: Define and Use Delegates \(C++/CLI\)](#)
- [How to: Define and consume enums in C++/CLI](#)
- [How to: Use Events in C++/CLI](#)
- [How to: Define an Interface Static Constructor \(C++/CLI\)](#)
- [How to: Declare Override Specifiers in Native Compilations \(C++/CLI\)](#)
- [How to: Use Properties in C++/CLI](#)
- [How to: Use safe\\_cast in C++/CLI](#)

## Related Sections

- [.NET Programming with C++/CLI \(Visual C++\)](#)

# How to: Create CLR Empty Projects

9/20/2022 • 2 minutes to read • [Edit Online](#)

To create a CLR empty project, use the **CLR Empty Project** template, which is available from the **New Project** dialog box.

## NOTE

The appearance of features in the IDE can depend on your active settings or edition, and might differ from those described in Help. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

## To create a CLR empty project

1. On the **File** menu, click **New**, and then click **Project**.

The **New Project** dialog box appears.

2. Under **Installed Templates**, click the **Visual C++** node; then click the **CLR** node. Choose the **CLR Empty Project** icon.
3. In the **Name** box, enter a unique name for your application.

## NOTE

You can also specify other project and solution settings from the **New Project** dialog box, but these settings are not required.

4. Click **OK**.

## See also

[C++ project types in Visual Studio](#)

[Debugging C++ projects](#)

# How to: Create CLR Console Applications (C++/CLI)

9/20/2022 • 4 minutes to read • [Edit Online](#)

You can use the **CLR Console Application** template in the **New Project** dialog to create a console app project that already has essential project references and files.

You can use the **CLR Console App** template in the **New Project** dialog to create a console app project that already has essential project references and files.

C++/CLI support isn't installed by default when you install a Visual Studio C++ workload. If you don't see a CLR heading under Visual C++ in the **New Project** dialog, you may need to install C++/CLI support. For more information, see [.NET programming with C++/CLI](#).

You can use the **CLR Console App (.NET Framework)** template in the **Create a new project** dialog to create a console app project that already has essential project references and files.

C++/CLI support isn't installed by default when you install a Visual Studio C++ workload. If you don't see CLR project templates in the **Create a new project** dialog, you may need to install C++/CLI support. For more information, see [.NET programming with C++/CLI](#).

Typically, a console app is compiled into a stand-alone executable file but doesn't have a graphical user interface. Users run the console app at a command prompt. They can use the command line to issue instructions to the running app. The app provides output information as text in the command window. The immediate feedback of a console app makes it a great way to learn programming. You don't need to worry about how to implement a graphical user interface.

When you use the CLR Console Application template to create a project, it automatically adds these references and files:

- References to these .NET Framework namespaces:
  - **System, System.Data, System.Xml**: These references contain the fundamental classes that define commonly used types, events, interfaces, attributes, and exceptions.
  - **mscorlib.dll** : The assembly DLL that supports .NET Framework development.
- Source files:
  - **ConsoleApplicationName.cpp** : The main source file and entry point into the app. This file has the base name you specified for your project. It identifies the project DLL file and the project namespace. Provide your own code in this file.
  - **AssemblyInfo.cpp** : Contains attributes and settings that you can use to modify the project's assembly metadata. For more information, see [Assembly contents](#).
  - **stdafx.cpp** : Used to build a precompiled header file that's named **ConsoleApplicationName.pch** and a precompiled types file that's named **stdafx.obj**.
- Header files:
  - **stdafx.h** : Used to build a precompiled header file that's named **ConsoleApplicationName.pch** and a precompiled types file that's named **stdafx.obj**.

- `resource.h` : A generated include file for `app.rc`.
- Resource files:
  - `app.rc` : The resource script file of a program.
  - `app.ico` : The icon file of a program.
- `ReadMe.txt` : Describes the files in the project.

When you use the CLR Console App template to create a project, it automatically adds these references and files:

- References to these .NET Framework namespaces:
  - `System, System.Data, System.Xml`: These references contain the fundamental classes that define commonly used types, events, interfaces, attributes, and exceptions.
  - `mscorlib.dll` : The assembly DLL that supports .NET Framework development.
- Source files:
  - `ConsoleApplicationName.cpp` : The main source file and entry point into the app. This file has the base name you specified for your project. It identifies the project DLL file and the project namespace. Provide your own code in this file.
  - `AssemblyInfo.cpp` : Contains attributes and settings that you can use to modify the project's assembly metadata. For more information, see [Assembly contents](#).
  - `pch.cpp` : Used to build a precompiled header file that's named `ConsoleApplicationName.pch` and a precompiled types file that's named `pch.obj`.
- Header files:
  - `pch.h` : Used to build a precompiled header file that's named `ConsoleApplicationName.pch` and a precompiled types file that's named `pch.obj`.
  - `Resource.h` : A generated include file for `app.rc`.
- Resource files:
  - `app.rc` : The resource script file of a program.
  - `app.ico` : The icon file of a program.

## To create a CLR console app project

1. On the menu bar, choose **File > New > Project**.
  2. In the **New Project** dialog box, select the **Installed > Templates > Visual C++ > CLR** node, and then select the **CLR Console Application** template.
  3. In the **Name** box, enter a unique name for your application.  
You can specify other project and solution settings, but they're not required.
  4. Choose the **OK** button to generate the project and source files.
1. On the menu bar, choose **File > New > Project**.
  2. In the **New Project** dialog box, select the **Installed > Visual C++ > CLR** node, and then select the **CLR Console App** template.

3. In the **Name** box, enter a unique name for your application.

You can specify other project and solution settings, but they're not required.

4. Choose the **OK** button to generate the project and source files.

1. On the menu bar, choose **File > New > Project**.

2. In the **Create a new project** dialog box, enter "clr console" in the search box. Select the **CLR Console App (.NET Framework)** template, and then choose **Next**.

3. In the **Name** box, enter a unique name for your application.

You can specify other project and solution settings, but they're not required.

4. Choose the **Create** button to generate the project and source files.

## See also

[CLR projects](#)

# How to: Use Tracking References in C++/CLI

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article shows how to use a tracking reference (%) in C++/CLI to pass common language runtime (CLR) types by reference.

## To pass CLR types by reference

The following sample shows how to use a tracking reference to pass CLR types by reference.

```
// tracking_reference_handles.cpp
// compile with: /clr
using namespace System;

ref struct City {
private:
    Int16 zip_;

public:
    City (int zip) : zip_(zip) {};
    property Int16 zip {
        Int16 get(void) {
            return zip_;
        } // get
    } // property
};

void passByRef (City ^% myCity) {
    // cast required so this pointer in City struct is "const City"
    if (myCity->zip == 20100)
        Console::WriteLine("zip == 20100");
    else
        Console::WriteLine("zip != 20100");
}

ref class G {
public:
    int i;
};

void Test(int % i) {
    i++;
}

int main() {
    G ^ g1 = gcnew G;
    G ^% g2 = g1;
    g1 -> i = 12;

    Test(g2->i); // g2->i will be changed in Test()

    City ^ Milano = gcnew City(20100);
    passByRef(Milano);
}
```

```
zip == 20100
```

The next sample shows that taking the address of a tracking reference returns an [interior\\_ptr \(C++/CLI\)](#), and

shows how to modify and access data through a tracking reference.

```
// tracking_reference_data.cpp
// compile with: /clr
using namespace System;

public ref class R {
public:
    R(int i) : m_i(i) {
        Console::WriteLine("ctor: R(int)");
    }

    int m_i;
};

class N {
public:
    N(int i) : m_i (i) {
        Console::WriteLine("ctor: N(int i)");
    }

    int m_i;
};

int main() {
    R ^hr = gcnew R('r');
    R ^%thr = hr;
    N n('n');
    N %tn = n;

    // Declare interior pointers
    interior_ptr<R^> iphr = &thr;
    interior_ptr<N> ipn = &tn;

    // Modify data through interior pointer
    (*iphr)->m_i = 1;    // (*iphr)->m_i == thr->m_i
    ipn->m_i = 4;    // ipn->m_i == tn.m_i

    ++thr-> m_i;    // hr->m_i == thr->m_i
    ++tn. m_i;    // n.m_i == tn.m_i

    ++hr-> m_i;    // (*iphr)->m_i == hr->m_i
    ++n. m_i;    // ipn->m_i == n.m_i
}
```

```
ctor: R(int)
ctor: N(int i)
```

## Tracking references and interior pointers

The following code sample shows that you can convert between tracking references and interior pointers.

```

// tracking_reference_interior_ptr.cpp
// compile with: /clr
using namespace System;

public ref class R {
public:
    R(int i) : m_i(i) {
        Console::WriteLine("ctor: R(int)");
    }

    int m_i;
};

class N {
public:
    N(int i) : m_i(i) {
        Console::WriteLine("ctor: N(int i)");
    }

    int m_i;
};

int main() {
    R ^hr = gcnew R('r');
    N n('n');

    R ^thr = hr;
    N ^tn = n;

    // Declare interior pointers
    interior_ptr<R^> iphr = &hr;
    interior_ptr<N> ipn = &n;

    // Modify data through interior pointer
    (*iphr)->m_i = 1;    // (*iphr)-> m_i == thr->m_i
    ipn->m_i = 4;    // ipn->m_i == tn.m_i

    ++thr->m_i;    // hr->m_i == thr->m_i
    ++tn.m_i;    // n.m_i == tn.m_i

    ++hr->m_i;    // (*iphr)->m_i == hr->m_i
    ++n.m_i;    // ipn->m_i == n.m_i
}

```

```

ctor: R(int)
ctor: N(int i)

```

## Tracking references and value types

This sample shows simple boxing through a tracking reference to a value type:

```

// tracking_reference_valuetypes_1.cpp
// compile with: /clr

using namespace System;

int main() {
    int i = 10;
    int % j = i;
    Object ^ o = j;    // j is implicitly boxed and assigned to o
}

```

The next sample shows that you can have both tracking references and native references to value types.

```
// tracking_reference_valuetypes_2.cpp
// compile with: /clr
using namespace System;
int main() {
    int i = 10;
    int & j = i;
    int % k = j;
    i++; // 11
    j++; // 12
    k++; // 13
    Console::WriteLine(i);
    Console::WriteLine(j);
    Console::WriteLine(k);
}
```

```
13
13
13
```

The following sample shows that you can use tracking references together with value types and native types.

```
// tracking_reference_valuetypes_3.cpp
// compile with: /clr
value struct G {
    int i;
};

struct H {
    int i;
};

int main() {
    G g;
    G % v = g;
    v.i = 4;
    System::Console::WriteLine(v.i);
    System::Console::WriteLine(g.i);

    H h;
    H % w = h;
    w.i = 5;
    System::Console::WriteLine(w.i);
    System::Console::WriteLine(h.i);
}
```

```
4
4
5
5
```

This sample shows that you can bind a tracking reference to a value type on the garbage-collected heap:

```

// tracking_reference_valuetypes_4.cpp
// compile with: /clr
using namespace System;
value struct V {
    int i;
};

void Test(V^ hv) { // hv boxes another copy of original V on GC heap
    Console::WriteLine("Boxed new copy V: {0}", hv->i);
}

int main() {
    V v; // V on the stack
    v.i = 1;
    V ^hv1 = v; // v is boxed and assigned to hv1
    v.i = 2;
    V % trV = *hv1; // trV is bound to boxed v, the v on the gc heap.
    Console::WriteLine("Original V: {0}, Tracking reference to boxed V: {1}", v.i, trV.i);
    V ^hv2 = trV; // hv2 boxes another copy of boxed v on the GC heap
    hv2->i = 3;
    Console::WriteLine("Tracking reference to boxed V: {0}", hv2->i);
    Test(trV);
    v.i = 4;
    V ^% trhV = hv1; // creates tracking reference to boxed type handle
    Console::WriteLine("Original V: {0}, Reference to handle of originally boxed V: {1}", v.i, trhV->i);
}

```

```

Original V: 2, Tracking reference to boxed V: 1
Tracking reference to boxed V: 3
Boxed new copy V: 1
Original V: 4, Reference to handle of originally boxed V: 1

```

## Template functions that take native, value, or reference parameters

By using a tracking reference in the signature of a template function, you ensure that the function can be called by a parameter whose type is native, CLR value, or CLR reference.

```

// tracking_reference_template.cpp
// compile with: /clr
using namespace System;

class Temp {
public:
    // template functions
    template<typename T>
    static int f1(T% tt) {    // works for object in any location
        Console::WriteLine("T %");
        return 0;
    }

    template<typename T>
    static int f2(T& rt) {    // won't work for object on the gc heap
        Console::WriteLine("T &");
        return 1;
    }
};

// Class Definitions
ref struct R {
    int i;
};

int main() {
    R ^hr = gcnew R;
    int i = 1;

    Temp::f1(i); // ok
    Temp::f1(hr->i); // ok
    Temp::f2(i); // ok

    // error can't track object on gc heap with a native reference
    // Temp::f2(hr->i);
}

```

```

T %
T %
T &

```

## See also

[Tracking Reference Operator](#)

# How to: Use Arrays in C++/CLI

9/20/2022 • 11 minutes to read • [Edit Online](#)

This article describes how to use arrays in C++/CLI.

## Single-dimension arrays

The following sample shows how to create single-dimension arrays of reference, value, and native pointer types. It also shows how to return a single-dimension array from a function and how to pass a single-dimension array as an argument to a function.

```
// mcppv2_sdarrays.cpp
// compile with: /clr
using namespace System;

#define ARRAY_SIZE 2

value struct MyStruct {
    int m_i;
};

ref class MyClass {
public:
    int m_i;
};

struct MyNativeClass {
    int m_i;
};

// Returns a managed array of a reference type.
array<MyClass^>^ Test0() {
    int i;
    array< MyClass^ >^ local = gcnew array< MyClass^ >(ARRAY_SIZE);

    for (i = 0 ; i < ARRAY_SIZE ; i++) {
        local[i] = gcnew MyClass;
        local[i] -> m_i = i;
    }
    return local;
}

// Returns a managed array of Int32.
array<Int32>^ Test1() {
    int i;
    array< Int32 >^ local = gcnew array< Int32 >(ARRAY_SIZE);

    for (i = 0 ; i < ARRAY_SIZE ; i++)
        local[i] = i + 10;
    return local;
}

// Modifies an array.
void Test2(array< MyNativeClass * >^ local) {
    for (int i = 0 ; i < ARRAY_SIZE ; i++)
        local[i] -> m_i = local[i] -> m_i + 2;
}

int main() {
    int i;
```

```

// Declares an array of user-defined reference types
// and uses a function to initialize.
array< MyClass^ >^ MyClass0;
MyClass0 = Test0();

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyClass0[{0}] = {1}", i, MyClass0[i] -> m_i);
Console::WriteLine();

// Declares an array of value types and uses a function to initialize.
array< Int32 >^ IntArray;
IntArray = Test1();

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("IntArray[{0}] = {1}", i, IntArray[i]);
Console::WriteLine();

// Declares and initializes an array of user-defined
// reference types.
array< MyClass^ >^ MyClass1 = gcnew array< MyClass^ >(ARRAY_SIZE);
for (i = 0 ; i < ARRAY_SIZE ; i++) {
    MyClass1[i] = gcnew MyClass;
    MyClass1[i] -> m_i = i + 20;
}

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyClass1[{0}] = {1}", i, MyClass1[i] -> m_i);
Console::WriteLine();

// Declares and initializes an array of pointers to a native type.
array< MyNativeClass * >^ MyClass2 = gcnew array<
    MyNativeClass * >(ARRAY_SIZE);
for (i = 0 ; i < ARRAY_SIZE ; i++) {
    MyClass2[i] = new MyNativeClass();
    MyClass2[i] -> m_i = i + 30;
}

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyClass2[{0}] = {1}", i, MyClass2[i]->m_i);
Console::WriteLine();

Test2(MyClass2);
for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyClass2[{0}] = {1}", i, MyClass2[i]->m_i);
Console::WriteLine();

delete[] MyClass2[0];
delete[] MyClass2[1];

// Declares and initializes an array of user-defined value types.
array< MyStruct >^ MyStruct1 = gcnew array< MyStruct >(ARRAY_SIZE);
for (i = 0 ; i < ARRAY_SIZE ; i++) {
    MyStruct1[i] = MyStruct();
    MyStruct1[i].m_i = i + 40;
}

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyStruct1[{0}] = {1}", i, MyStruct1[i].m_i);
}

```

```

 MyClass0[0] = 0
 MyClass0[1] = 1

 IntArray[0] = 10
 IntArray[1] = 11

 MyClass1[0] = 20
 MyClass1[1] = 21

 MyClass2[0] = 30
 MyClass2[1] = 31

 MyClass2[0] = 32
 MyClass2[1] = 33

 MyStruct1[0] = 40
 MyStruct1[1] = 41

```

The next sample shows how to perform aggregate initialization on single-dimension managed arrays.

```

// mcppv2_sdarrays_aggregate_init.cpp
// compile with: /clr
using namespace System;

ref class G {
public:
    G(int i) {}
};

value class V {
public:
    V(int i) {}
};

class N {
public:
    N(int i) {}
};

int main() {
    // Aggregate initialize a single-dimension managed array.
    array<String^>^ gc1 = gcnew array<String^>{"one", "two", "three"};
    array<String^>^ gc2 = {"one", "two", "three"};

    array<G^>^ gc3 = gcnew array<G>{gcnew G(0), gcnew G(1), gcnew G(2)};
    array<G^>^ gc4 = {gcnew G(0), gcnew G(1), gcnew G(2)};

    array<Int32>^ value1 = gcnew array<Int32>{0, 1, 2};
    array<Int32>^ value2 = {0, 1, 2};

    array<V>^ value3 = gcnew array<V>{V(0), V(1), V(2)};
    array<V>^ value4 = {V(0), V(1), V(2)};

    array<N*>^ native1 = gcnew array<N*>{new N(0), new N(1), new N(2)};
    array<N*>^ native2 = {new N(0), new N(1), new N(2)};
}

```

```

MyClass0[0, 0] = 0
MyClass0[0, 1] = 0
MyClass0[1, 0] = 1
MyClass0[1, 1] = 1

IntArray[0, 0] = 10
IntArray[0, 1] = 10
IntArray[1, 0] = 11
IntArray[1, 1] = 11

```

This example shows how to perform aggregate initialization on a multi-dimension managed array:

```

// mcppv2_mdarrays_aggregate_initialization.cpp
// compile with: /clr
using namespace System;

ref class G {
public:
    G(int i) {}
};

value class V {
public:
    V(int i) {}
};

class N {
public:
    N(int i) {}
};

int main() {
    // Aggregate initialize a multidimension managed array.
    array<String^, 2>^ gc1 = gcnew array<String^, 2>{ {"one", "two"}, 
        {"three", "four"} };
    array<String^, 2>^ gc2 = { {"one", "two"}, {"three", "four"} };

    array<G^, 2>^ gc3 = gcnew array<G^, 2>{ {gcnew G(0), gcnew G(1)}, 
        {gcnew G(2), gcnew G(3)} };
    array<G^, 2>^ gc4 = { {gcnew G(0), gcnew G(1)}, {gcnew G(2), gcnew G(3)} };

    array<Int32, 2>^ value1 = gcnew array<Int32, 2>{ {0, 1}, {2, 3} };
    array<Int32, 2>^ value2 = { {0, 1}, {2, 3} };

    array<V, 2>^ value3 = gcnew array<V, 2>{ {V(0), V(1)}, {V(2), V(3)} };
    array<V, 2>^ value4 = { {V(0), V(1)}, {V(2), V(3)} };

    array<N*, 2>^ native1 = gcnew array<N*, 2>{ {new N(0), new N(1)}, 
        {new N(2), new N(3)} };
    array<N*, 2>^ native2 = { {new N(0), new N(1)}, {new N(2), new N(3)} };
}

```

## Jagged arrays

This section shows how to create single-dimension arrays of managed arrays of reference, value, and native pointer types. It also shows how to return a single-dimension array of managed arrays from a function and how to pass a single-dimension array as an argument to a function.

```

// mcppv2_array_of_arrays.cpp
// compile with: /clr
using namespace System;

#define ARRAY_SIZE 2

```

```

#INCLUDE ARRAY_SIZE 2

value struct MyStruct {
    int m_i;
};

ref class MyClass {
public:
    int m_i;
};

// Returns an array of managed arrays of a reference type.
array<array<MyClass^>^>^ Test0() {
    int size_of_array = 4;
    array<array<MyClass^>^>^ local = gcnew
        array<array<MyClass^>^>(ARRAY_SIZE);

    for (int i = 0 ; i < ARRAY_SIZE ; i++, size_of_array += 4) {
        local[i] = gcnew array<MyClass^>(size_of_array);
        for (int k = 0; k < size_of_array ; k++) {
            local[i][k] = gcnew MyClass;
            local[i][k] -> m_i = i;
        }
    }

    return local;
}

// Returns a managed array of Int32.
array<array<Int32>^>^ Test1() {
    int i;
    array<array<Int32>^>^ local = gcnew array<array< Int32 >^>(ARRAY_SIZE);

    for (i = 0 ; i < ARRAY_SIZE ; i++) {
        local[i] = gcnew array< Int32 >(ARRAY_SIZE);
        for ( int j = 0 ; j < ARRAY_SIZE ; j++ )
            local[i][j] = i + 10;
    }
    return local;
}

int main() {
    int i, j;

    // Declares an array of user-defined reference types
    // and uses a function to initialize.
    array< array< MyClass^ >^ >^ MyClass0;
    MyClass0 = Test0();

    for (i = 0 ; i < ARRAY_SIZE ; i++)
        for ( j = 0 ; j < ARRAY_SIZE ; j++ )
            Console::WriteLine("MyClass0[{0}] = {1}", i, MyClass0[i][j] -> m_i);
    Console::WriteLine();

    // Declares an array of value types and uses a function to initialize.
    array< array< Int32 >^ >^ IntArray;
    IntArray = Test1();

    for (i = 0 ; i < ARRAY_SIZE ; i++)
        for (j = 0 ; j < ARRAY_SIZE ; j++)
            Console::WriteLine("IntArray[{0}] = {1}", i, IntArray[i][j]);
    Console::WriteLine();

    // Declares and initializes an array of user-defined value types.
    array< MyStruct >^ MyStruct1 = gcnew array< MyStruct >(ARRAY_SIZE);
    for (i = 0 ; i < ARRAY_SIZE ; i++) {
        MyStruct1[i] = MyStruct();
        MyStruct1[i].m_i = i + 40;
    }
}

```

```

    for (i = 0 ; i < ARRAY_SIZE ; i++)
        Console::WriteLine(MyStruct1[i].m_i);
}

```

```

 MyClass0[0] = 0
 MyClass0[0] = 0
 MyClass0[1] = 1
 MyClass0[1] = 1

 IntArray[0] = 10
 IntArray[0] = 10
 IntArray[1] = 11
 IntArray[1] = 11

40
41

```

The following sample shows how to perform aggregate initialization with jagged arrays.

```

// mcppv2_array_of_arrays_aggregate_init.cpp
// compile with: /clr
using namespace System;
#define ARRAY_SIZE 2
int size_of_array = 4;
int count = 0;

ref class MyClass {
public:
    int m_i;
};

struct MyNativeClass {
    int m_i;
};

int main() {
    // Declares an array of user-defined reference types
    // and performs an aggregate initialization.
    array< array< MyClass^ >^ >^ MyClass0 = gcnew array< array< MyClass^ >^ > {
        gcnew array< MyClass^ >{ gcnew MyClass(), gcnew MyClass() },
        gcnew array< MyClass^ >{ gcnew MyClass(), gcnew MyClass() }
    };

    for ( int i = 0 ; i < ARRAY_SIZE ; i++, size_of_array += 4 )
        for ( int k = 0 ; k < ARRAY_SIZE ; k++ )
            MyClass0[i][k] -> m_i = i;

    for ( int i = 0 ; i < ARRAY_SIZE ; i++ )
        for ( int j = 0 ; j < ARRAY_SIZE ; j++ )
            Console::WriteLine("MyClass0[{0}] = {1}", i, MyClass0[i][j] -> m_i);
    Console::WriteLine();

    // Declares an array of value types and performs an aggregate initialization.
    array< array< Int32 >^ >^ IntArray = gcnew array< array< Int32 >^ > {
        gcnew array< Int32 >{1,2},
        gcnew array< Int32 >{3,4,5}
    };

    for each ( array< int>^ outer in IntArray ) {
        Console::Write("[");
        for each( int i in outer )
            Console::Write(" {0}", i);
        Console::Write(" ]");
    }
}

```

```

Console::WriteLine();
}

Console::WriteLine();

// Declares and initializes an array of pointers to a native type.
array<array< MyNativeClass * >^ > ^ MyClass2 =
    gcnew array<array< MyNativeClass * > ^ > {
        gcnew array<MyNativeClass *>{ new MyNativeClass(), new MyNativeClass() },
        gcnew array<MyNativeClass *>{ new MyNativeClass(), new MyNativeClass(), new MyNativeClass() }
    };

for each ( array<MyNativeClass *> ^ outer in MyClass2 )
    for each( MyNativeClass* i in outer )
        i->m_i = count++;

for each ( array<MyNativeClass *> ^ outer in MyClass2 ) {
    Console::Write("[");
    for each( MyNativeClass* i in outer )
        Console::Write(" {0}", i->m_i);
    Console::Write("]");
    Console::WriteLine();
}
Console::WriteLine();

// Declares and initializes an array of two-dimensional arrays of strings.
array<array<String ^,2> ^> ^gc3 = gcnew array<array<String ^,2> ^>{
    gcnew array<String ^>{ {"a","b"}, {"c", "d"}, {"e","f"} },
    gcnew array<String ^>{ {"g", "h"} }
};

for each ( array<String^, 2> ^ outer in gc3 ){
    Console::Write("[");
    for each( String ^ i in outer )
        Console::Write(" {0}", i);
    Console::Write("]");
    Console::WriteLine();
}
}
}

```

```

MyClass0[0] = 0
MyClass0[0] = 0
MyClass0[1] = 1
MyClass0[1] = 1

[ 1 2 ]
[ 3 4 5 ]

[ 0 1 ]
[ 2 3 4 ]

[ a b c d e f ]
[ g h ]

```

## Managed arrays as template type parameters

This example shows how to use a managed array as a parameter to a template:

```

// mcppv2_template_type_params.cpp
// compile with: /clr
using namespace System;
template <class T>
class TA {
public:
    array<array<T>^>^ f() {
        array<array<T>^>^ larr = gcnew array<array<T>^>(10);
        return larr;
    }
};

int main() {
    int retval = 0;
    TA<array<array<Int32>^>^* ta1 = new TA<array<array<Int32>^>^>();
    array<array<array<Int32>^>^>^ larr = ta1->f();
    retval += larr->Length - 10;
    Console::WriteLine("Return Code: {0}", retval);
}

```

Return Code: 0

## typedefs for managed arrays

This example shows how to make a typedef for a managed array:

```

// mcppv2_typedef_arrays.cpp
// compile with: /clr
using namespace System;
ref class G {};

typedef array<array<G^>^> jagged_array;

int main() {
    jagged_array ^ MyArr = gcnew jagged_array (10);
}

```

## Sorting arrays

Unlike standard C++ arrays, managed arrays are implicitly derived from an array base class from which they inherit common behavior. An example is the `Sort` method, which can be used to order the items in any array.

For arrays that contain basic intrinsic types, you can call the `Sort` method. You can override the sort criteria, and doing so is required when you want to sort for arrays of complex types. In this case, the array element type must implement the `CompareTo` method.

```

// array_sort.cpp
// compile with: /clr
using namespace System;

int main() {
    array<int>^ a = { 5, 4, 1, 3, 2 };
    Array::Sort( a );
    for (int i=0; i < a->Length; i++)
        Console::Write("{0} ", a[i] );
}

```

## Sorting arrays by using custom criteria

To sort arrays that contain basic intrinsic types, just call the `Array::Sort` method. However, to sort arrays that contain complex types or to override the default sort criteria, override the `CompareTo` method.

In the following example, a structure named `Element` is derived from `IComparable`, and written to provide a `CompareTo` method that uses the average of two integers as the sort criterion.

```
using namespace System;

value struct Element : public IComparable {
    int v1, v2;

    virtual int CompareTo(Object^ obj) {
        Element^ o = dynamic_cast<Element^>(obj);
        if (o) {
            int thisAverage = (v1 + v2) / 2;
            int thatAverage = (o->v1 + o->v2) / 2;
            if (thisAverage < thatAverage)
                return -1;
            else if (thisAverage > thatAverage)
                return 1;
            return 0;
        }
        else
            throw gcnew ArgumentException
            ("Object must be of type 'Element'");
    }
};

int main() {
    array<Element>^ a = gcnew array<Element>(10);
    Random^ r = gcnew Random;

    for (int i=0; i < a->Length; i++) {
        a[i].v1 = r->Next() % 100;
        a[i].v2 = r->Next() % 100;
    }

    Array::Sort( a );
    for (int i=0; i < a->Length; i++) {
        int v1 = a[i].v1;
        int v2 = a[i].v2;
        int v = (v1 + v2) / 2;
        Console::WriteLine("{0} ({1}+{2})/2 ", v, v1, v2);
    }
}
```

## Array covariance

Given reference class D that has direct or indirect base class B, an array of type D can be assigned to an array variable of type B.

```
// clr_array_covariance.cpp
// compile with: /clr
using namespace System;

int main() {
    // String derives from Object.
    array<Object^>^ oa = gcnew array<String^>(20);
}
```

An assignment to an array element shall be assignment-compatible with the dynamic type of the array. An assignment to an array element that has an incompatible type causes `System::ArrayTypeMismatchException` to be thrown.

Array covariance doesn't apply to arrays of value class type. For example, arrays of `Int32` cannot be converted to `Object^` arrays, not even by using boxing.

```
// clr_array_covariance2.cpp
// compile with: /clr
using namespace System;

ref struct Base { int i; };
ref struct Derived : Base {};
ref struct Derived2 : Base {};
ref struct Derived3 : Derived {};
ref struct Other { short s; };

int main() {
    // Derived* d[] = new Derived*[100];
    array<Derived^> ^ d = gcnew array<Derived^>(100);

    // ok by array covariance
    array<Base ^> ^ b = d;

    // invalid
    // b[0] = new Other;

    // error (runtime exception)
    // b[1] = gcnew Derived2;

    // error (runtime exception),
    // must be "at least" a Derived.
    // b[0] = gcnew Base;

    b[1] = gcnew Derived;
    b[0] = gcnew Derived3;
}
```

## See also

[Arrays](#)

# How to: Define and consume classes and structs (C++/CLI)

9/20/2022 • 19 minutes to read • [Edit Online](#)

This article shows how to define and consume user-defined reference types and value types in C++/CLI.

## Object instantiation

Reference (ref) types can only be instantiated on the managed heap, not on the stack or on the native heap.  
Value types can be instantiated on the stack or the managed heap.

```

// mcppv2_ref_class2.cpp
// compile with: /clr
ref class MyClass {
public:
    int i;

    // nested class
    ref class MyClass2 {
    public:
        int i;
    };

    // nested interface
    interface struct MyInterface {
        void f();
    };
};

ref class MyClass2 : public MyClass::MyInterface {
public:
    virtual void f() {
        System::Console::WriteLine("test");
    }
};

public value struct MyStruct {
    void f() {
        System::Console::WriteLine("test");
    }
};

int main() {
    // instantiate ref type on garbage-collected heap
    MyClass ^ p_MyClass = gcnew MyClass;
    p_MyClass -> i = 4;

    // instantiate value type on garbage-collected heap
    MyStruct ^ p_MyStruct = gcnew MyStruct;
    p_MyStruct -> f();

    // instantiate value type on the stack
    MyStruct p_MyStruct2;
    p_MyStruct2.f();

    // instantiate nested ref type on garbage-collected heap
    MyClass::MyClass2 ^ p_MyClass2 = gcnew MyClass::MyClass2;
    p_MyClass2 -> i = 5;
}

```

## Implicitly abstract classes

An *implicitly abstract class* can't be instantiated. A class is implicitly abstract when:

- the base type of the class is an interface, and
- the class doesn't implement all of the interface's member functions.

You may be unable to construct objects from a class that's derived from an interface. The reason might be that the class is implicitly abstract. For more information about abstract classes, see [abstract](#).

The following code example demonstrates that the `MyClass` class can't be instantiated because function `MyClass::func2` isn't implemented. To enable the example to compile, uncomment `MyClass::func2`.

```
// mcppv2_ref_class5.cpp
// compile with: /clr
interface struct MyInterface {
    void func1();
    void func2();
};

ref class MyClass : public MyInterface {
public:
    void func1(){}
    // void func2(){}
};

int main() {
    MyClass ^ h_MyClass = gcnew MyClass;    // C2259
                                            // To resolve, uncomment MyClass::func2.
}
```

## Type visibility

You can control the visibility of common language runtime (CLR) types. When your assembly is referenced, you control whether types in the assembly are visible or not visible outside the assembly.

`public` indicates that a type is visible to any source file that contains a `#using` directive for the assembly that contains the type. `private` indicates that a type isn't visible to source files that contain a `#using` directive for the assembly that contains the type. However, private types are visible within the same assembly. By default, the visibility for a class is `private`.

By default before Visual Studio 2005, native types had public accessibility outside the assembly. Enable [Compiler Warning \(level 1\) C4692](#) to help you see where private native types are used incorrectly. Use the `make_public` pragma to give public accessibility to a native type in a source code file that you can't modify.

For more information, see [#using Directive](#).

The following sample shows how to declare types and specify their accessibility, and then access those types inside the assembly. If an assembly that has private types is referenced by using `#using`, only public types in the assembly are visible.

```

// type_visibility.cpp
// compile with: /clr
using namespace System;
// public type, visible inside and outside assembly
public ref struct Public_Class {
    void Test(){Console::WriteLine("in Public_Class");}
};

// private type, visible inside but not outside assembly
private ref struct Private_Class {
    void Test(){Console::WriteLine("in Private_Class");}
};

// default accessibility is private
ref class Private_Class_2 {
public:
    void Test(){Console::WriteLine("in Private_Class_2");}
};

int main() {
    Public_Class ^ a = gcnew Public_Class;
    a->Test();

    Private_Class ^ b = gcnew Private_Class;
    b->Test();

    Private_Class_2 ^ c = gcnew Private_Class_2;
    c->Test();
}

```

## Output

```

in Public_Class
in Private_Class
in Private_Class_2

```

Now, let's rewrite the previous sample so that it's built as a DLL.

```

// type_visibility_2.cpp
// compile with: /clr /LD
using namespace System;
// public type, visible inside and outside the assembly
public ref struct Public_Class {
    void Test(){Console::WriteLine("in Public_Class");}
};

// private type, visible inside but not outside the assembly
private ref struct Private_Class {
    void Test(){Console::WriteLine("in Private_Class");}
};

// by default, accessibility is private
ref class Private_Class_2 {
public:
    void Test(){Console::WriteLine("in Private_Class_2");}
};

```

The next sample shows how to access types outside the assembly. In this sample, the client consumes the component that's built in the previous sample.

```

// type_visibility_3.cpp
// compile with: /clr
#using "type_visibility_2.dll"
int main() {
    Public_Class ^ a = gcnew Public_Class;
    a->Test();

    // private types not accessible outside the assembly
    // Private_Class ^ b = gcnew Private_Class;
    // Private_Class_2 ^ c = gcnew Private_Class_2;
}

```

## Output

```
in Public_Class
```

## Member visibility

You can make access to a member of a public class from within the same assembly different than access to it from outside the assembly by using pairs of the access specifiers `public`, `protected`, and `private`.

This table summarizes the effect of the various access specifiers:

SPECIFIER	EFFECT
<code>public</code>	Member is accessible inside and outside the assembly. For more information, see <a href="#">public</a> .
<code>private</code>	Member is inaccessible, both inside and outside the assembly. For more information, see <a href="#">private</a> .
<code>protected</code>	Member is accessible inside and outside the assembly, but only to derived types. For more information, see <a href="#">protected</a> .
<code>internal</code>	Member is public inside the assembly but private outside the assembly. <code>internal</code> is a context-sensitive keyword. For more information, see <a href="#">Context-Sensitive Keywords</a> .
<code>public protected</code> -or- <code>protected public</code>	Member is public inside the assembly but protected outside the assembly.
<code>private protected</code> -or- <code>protected private</code>	Member is protected inside the assembly but private outside the assembly.

The following sample shows a public type that has members that are declared using the different access specifiers. Then, it shows access to those members from inside the assembly.

```

// compile with: /clr
using namespace System;
// public type, visible inside and outside the assembly
public ref class Public_Class {
public:
    void Public_Function(){System::Console::WriteLine("in Public_Function");}
private:
    void Private_Function(){System::Console::WriteLine("in Private_Function");}
protected:
    void Protected_Function(){System::Console::WriteLine("in Protected_Function");}
internal:
    void Internal_Function(){System::Console::WriteLine("in Internal_Function");}
protected public:
    void Protected_Public_Function(){System::Console::WriteLine("in Protected_Public_Function");}
public protected:
    void Public_Protected_Function(){System::Console::WriteLine("in Public_Protected_Function");}
private protected:
    void Private_Protected_Function(){System::Console::WriteLine("in Private_Protected_Function");}
protected private:
    void Protected_Private_Function(){System::Console::WriteLine("in Protected_Private_Function");}
};

// a derived type, calls protected functions
ref struct MyClass : public Public_Class {
    void Test() {
        Console::WriteLine("=====");
        Console::WriteLine("in function of derived class");
        Protected_Function();
        Protected_Private_Function();
        Private_Protected_Function();
        Console::WriteLine("exiting function of derived class");
        Console::WriteLine("=====");
    }
};

int main() {
    Public_Class ^ a = gcnew Public_Class;
    MyClass ^ b = gcnew MyClass;
    a->Public_Function();
    a->Protected_Public_Function();
    a->Public_Protected_Function();

    // accessible inside but not outside the assembly
    a->Internal_Function();

    // call protected functions
    b->Test();

    // not accessible inside or outside the assembly
    // a->Private_Function();
}

```

## Output

```

in Public_Function
in Protected_Public_Function
in Public_Protected_Function
in Internal_Function
=====
in function of derived class
in Protected_Function
in Protected_Private_Function
in Private_Protected_Function
exiting function of derived class
=====

```

Now let's build the previous sample as a DLL.

```

// compile with: /clr /LD
using namespace System;
// public type, visible inside and outside the assembly
public ref class Public_Class {
public:
    void Public_Function(){System::Console::WriteLine("in Public_Function");}

private:
    void Private_Function(){System::Console::WriteLine("in Private_Function");}

protected:
    void Protected_Function(){System::Console::WriteLine("in Protected_Function");}

internal:
    void Internal_Function(){System::Console::WriteLine("in Internal_Function");}

protected public:
    void Protected_Public_Function(){System::Console::WriteLine("in Protected_Public_Function");}

public protected:
    void Public_Protected_Function(){System::Console::WriteLine("in Public_Protected_Function");}

private protected:
    void Private_Protected_Function(){System::Console::WriteLine("in Private_Protected_Function");}

protected private:
    void Protected_Private_Function(){System::Console::WriteLine("in Protected_Private_Function");}
};

// a derived type, calls protected functions
ref struct MyClass : public Public_Class {
    void Test() {
        Console::WriteLine("=====");
        Console::WriteLine("in function of derived class");
        Protected_Function();
        Protected_Private_Function();
        Private_Protected_Function();
        Console::WriteLine("exiting function of derived class");
        Console::WriteLine("=====");
    }
};

```

The following sample consumes the component that's created in the previous sample. It shows how to access the members from outside the assembly.

```

// compile with: /clr
#using "type_member_visibility_2.dll"
using namespace System;
// a derived type, calls protected functions
ref struct MyClass : public Public_Class {
    void Test() {
        Console::WriteLine("=====");
        Console::WriteLine("in function of derived class");
        Protected_Function();
        Protected_Public_Function();
        Public_Protected_Function();
        Console::WriteLine("exiting function of derived class");
        Console::WriteLine("=====");
    }
};

int main() {
    Public_Class ^ a = gcnew Public_Class;
    MyClass ^ b = gcnew MyClass;
    a->Public_Function();

    // call protected functions
    b->Test();

    // can't be called outside the assembly
    // a->Private_Function();
    // a->Internal_Function();
    // a->Protected_Private_Function();
    // a->Private_Protected_Function();
}

```

## Output

```

in Public_Function
=====
in function of derived class
in Protected_Function
in Protected_Public_Function
in Public_Protected_Function
exiting function of derived class
=====

```

## Public and private native classes

A native type can be referenced from a managed type. For example, a function in a managed type can take a parameter whose type is a native struct. If the managed type and function are public in an assembly, then the native type must also be public.

```

// native type
public struct N {
    N(){}
    int i;
}

```

Next, create the source code file that consumes the native type:

```
// compile with: /clr /LD
#include "mcppv2_ref_class3.h"
// public managed type
public ref struct R {
    // public function that takes a native type
    void f(N nn) {}
};
```

Now, compile a client:

```
// compile with: /clr
#using "mcppv2_ref_class3.dll"

#include "mcppv2_ref_class3.h"

int main() {
    R ^r = gcnew R;
    N n;
    r->f(n);
}
```

## Static constructors

A CLR type—for example, a class or struct—can have a static constructor that can be used to initialize static data members. A static constructor is called at most once, and is called before any static member of the type is accessed the first time.

An instance constructor always runs after a static constructor.

The compiler can't inline a call to a constructor if the class has a static constructor. The compiler can't inline a call to any member function if the class is a value type, has a static constructor, and doesn't have an instance constructor. The CLR may inline the call, but the compiler can't.

Define a static constructor as a private member function, because it's meant to be called only by the CLR.

For more information about static constructors, see [How to: Define an Interface Static Constructor \(C++/CLI\)](#).

```
// compile with: /clr
using namespace System;

ref class MyClass {
private:
    static int i = 0;

    static MyClass() {
        Console::WriteLine("in static constructor");
        i = 9;
    }

public:
    static void Test() {
        i++;
        Console::WriteLine(i);
    }
};

int main() {
    MyClass::Test();
    MyClass::Test();
}
```

## Output

```
in static constructor  
10  
11
```

## Semantics of the `this` pointer

When you're using C++\CLI to define types, the `this` pointer in a reference type is of type *handle*. The `this` pointer in a value type is of type *interior pointer*.

These different semantics of the `this` pointer can cause unexpected behavior when a default indexer is called. The next example shows the correct way to access a default indexer in both a ref type and a value type.

For more information, see [Handle to Object Operator \(^\)](#) and [interior\\_ptr \(C++/CLI\)](#)

```
// compile with: /clr  
using namespace System;  
  
ref struct A {  
    property Double default[Double] {  
        Double get(Double data) {  
            return data*data;  
        }  
    }  
  
    A() {  
        // accessing default indexer  
        Console::WriteLine("{0}", this[3.3]);  
    }  
};  
  
value struct B {  
    property Double default[Double] {  
        Double get(Double data) {  
            return data*data;  
        }  
    }  
    void Test() {  
        // accessing default indexer  
        Console::WriteLine("{0}", this->default[3.3]);  
    }  
};  
  
int main() {  
    A ^ mya = gcnew A();  
    B ^ myb = gcnew B();  
    myb->Test();  
}
```

## Output

```
10.89  
10.89
```

## Hide-by-signature functions

In standard C++, a function in a base class gets hidden by a function that has the same name in a derived class, even if the derived-class function doesn't have the same kind or number of parameters. It's known as *hide-by-signature*.

*name* semantics. In a reference type, a function in a base class only gets hidden by a function in a derived class if both the name and the parameter list are the same. It's known as *hide-by-signature* semantics.

A class is considered a hide-by-signature class when all of its functions are marked in the metadata as `hidebysig`. By default, all classes that are created under `/clr` have `hidebysig` functions. When a class has `hidebysig` functions, the compiler doesn't hide functions by name in any direct base classes, but if the compiler encounters a hide-by-name class in an inheritance chain, it continues that hide-by-name behavior.

Under hide-by-signature semantics, when a function is called on an object, the compiler identifies the most derived class that contains a function that could satisfy the function call. If there's only one function in the class that satisfies the call, the compiler calls that function. If there's more than one function in the class that could satisfy the call, the compiler uses overload resolution rules to determine which function to call. For more information about overload rules, see [Function Overloading](#).

For a given function call, a function in a base class might have a signature that makes it a slightly better match than a function in a derived class. However, if the function was explicitly called on an object of the derived class, the function in the derived class is called.

Because the return value isn't considered part of a function's signature, a base-class function gets hidden if it has the same name and takes the same kind and number of arguments as a derived-class function, even if it differs in the type of the return value.

The following sample shows that a function in a base class isn't hidden by a function in a derived class.

```
// compile with: /clr
using namespace System;
ref struct Base {
    void Test() {
        Console::WriteLine("Base::Test");
    }
};

ref struct Derived : public Base {
    void Test(int i) {
        Console::WriteLine("Derived::Test");
    }
};

int main() {
    Derived ^ t = gcnew Derived;
    // Test() in the base class will not be hidden
    t->Test();
}
```

## Output

```
Base::Test
```

The next sample shows that the Microsoft C++ compiler calls a function in the most derived class—even if a conversion is required to match one or more of the parameters—and not call a function in a base class that is a better match for the function call.

```

// compile with: /clr
using namespace System;
ref struct Base {
    void Test2(Single d) {
        Console::WriteLine("Base::Test2");
    }
};

ref struct Derived : public Base {
    void Test2(Double f) {
        Console::WriteLine("Derived::Test2");
    }
};

int main() {
    Derived ^ t = gcnew Derived;
    // Base::Test2 is a better match, but the compiler
    // calls a function in the derived class if possible
    t->Test2(3.14f);
}

```

## Output

```
Derived::Test2
```

The following sample shows that it's possible to hide a function even if the base class has the same signature as the derived class.

```

// compile with: /clr
using namespace System;
ref struct Base {
    int Test4() {
        Console::WriteLine("Base::Test4");
        return 9;
    }
};

ref struct Derived : public Base {
    char Test4() {
        Console::WriteLine("Derived::Test4");
        return 'a';
    }
};

int main() {
    Derived ^ t = gcnew Derived;

    // Base::Test4 is hidden
    int i = t->Test4();
    Console::WriteLine(i);
}

```

## Output

```
Derived::Test4
97
```

## Copy constructors

The C++ standard says that a copy constructor is called when an object is moved, such that an object is created

and destroyed at the same address.

However, when a function that's compiled to MSIL calls a native function where a native class—or more than one—is passed by value and where the native class has a copy constructor or a destructor, no copy constructor is called and the object is destroyed at a different address than where it was created. This behavior could cause problems if the class has a pointer into itself, or if the code is tracking objects by address.

For more information, see [/clr \(Common Language Runtime Compilation\)](#).

The following sample demonstrates when a copy constructor isn't generated.

```
// compile with: /clr
#include<stdio.h>

struct S {
    int i;
    static int n;

    S() : i(n++) {
        printf_s("S object %d being constructed, this=%p\n", i, this);
    }

    S(S const& rhs) : i(n++) {
        printf_s("S object %d being copy constructed from S object "
                "%d, this=%p\n", i, rhs.i, this);
    }

    ~S() {
        printf_s("S object %d being destroyed, this=%p\n", i, this);
    }
};

int S::n = 0;

#pragma managed(push,off)
void f(S s1, S s2) {
    printf_s("in function f\n");
}
#pragma managed(pop)

int main() {
    S s;
    S t;
    f(s,t);
}
```

## Output

```
S object 0 being constructed, this=0018F378
S object 1 being constructed, this=0018F37C
S object 2 being copy constructed from S object 1, this=0018F380
S object 3 being copy constructed from S object 0, this=0018F384
S object 4 being copy constructed from S object 2, this=0018F2E4
S object 2 being destroyed, this=0018F380
S object 5 being copy constructed from S object 3, this=0018F2E0
S object 3 being destroyed, this=0018F384
in function f
S object 5 being destroyed, this=0018F2E0
S object 4 being destroyed, this=0018F2E4
S object 1 being destroyed, this=0018F37C
S object 0 being destroyed, this=0018F378
```

## Destructors and finalizers

Destructors in a reference type do a deterministic clean-up of resources. Finalizers clean up unmanaged resources, and can be called either deterministically by the destructor or nondeterministically by the garbage collector. For information about destructors in standard C++, see [Destructors](#).

```
class classname {  
    ~classname() {}    // destructor  
    ! classname() {}   // finalizer  
};
```

The CLR garbage collector deletes unused managed objects and releases their memory when they're no longer required. However, a type may use resources that the garbage collector doesn't know how to release. These resources are known as *unmanaged* resources (native file handles, for example). We recommend you release all unmanaged resources in the finalizer. The garbage collector releases managed resources nondeterministically, so it's not safe to refer to managed resources in a finalizer. That's because it's possible the garbage collector has already cleaned them up.

A Visual C++ finalizer isn't the same as the [Finalize](#) method. (CLR documentation uses finalizer and the [Finalize](#) method synonymously). The [Finalize](#) method is called by the garbage collector, which invokes each finalizer in a class inheritance chain. Unlike Visual C++ destructors, a derived-class finalizer call doesn't cause the compiler to invoke the finalizer in all base classes.

Because the Microsoft C++ compiler supports deterministic release of resources, don't try to implement the [Dispose](#) or [Finalize](#) methods. However, if you're familiar with these methods, here's how a Visual C++ finalizer and a destructor that calls the finalizer map to the [Dispose](#) pattern:

```
// Visual C++ code  
ref class T {  
    ~T() { this->!T(); }    // destructor calls finalizer  
    !T() {}    // finalizer  
};  
  
// equivalent to the Dispose pattern  
void Dispose(bool disposing) {  
    if (disposing) {  
        ~T();  
    } else {  
        !T();  
    }  
}
```

A managed type may also use managed resources that you'd prefer to release deterministically. You may not want the garbage collector to release an object nondeterministically at some point after the object is no longer required. The deterministic release of resources can significantly improve performance.

The Microsoft C++ compiler enables the definition of a destructor to deterministically clean up objects. Use the destructor to release all resources that you want to deterministically release. If a finalizer is present, call it from the destructor, to avoid code duplication.

```

// compile with: /clr /c
ref struct A {
    // destructor cleans up all resources
    ~A() {
        // clean up code to release managed resource
        // ...
        // to avoid code duplication,
        // call finalizer to release unmanaged resources
        this->!A();
    }

    // finalizer cleans up unmanaged resources
    // destructor or garbage collector will
    // clean up managed resources
    !A() {
        // clean up code to release unmanaged resources
        // ...
    }
};


```

If the code that consumes your type doesn't call the destructor, the garbage collector eventually releases all managed resources.

The presence of a destructor doesn't imply the presence of a finalizer. However, the presence of a finalizer implies that you must define a destructor and call the finalizer from that destructor. This call provides for the deterministic release of unmanaged resources.

Calling the destructor suppresses—by using [SuppressFinalize](#)—finalization of the object. If the destructor isn't called, your type's finalizer will eventually be called by the garbage collector.

You can improve performance by calling the destructor to deterministically clean up your object's resources, instead of letting the CLR nondeterministically finalize the object.

Code that's written in Visual C++ and compiled by using `/clr` runs a type's destructor if:

- An object that's created by using stack semantics goes out of scope. For more information, see [C++ Stack Semantics for Reference Types](#).
- An exception is thrown during the object's construction.
- The object is a member in an object whose destructor is running.
- You call the `delete` operator on a handle ([Handle to Object Operator \(^\)](#)).
- You explicitly call the destructor.

If a client that's written in another language consumes your type, the destructor gets called as follows:

- On a call to [Dispose](#).
- On a call to `Dispose(void)` on the type.
- If the type goes out of scope in a C# `using` statement.

If you're not using stack semantics for reference types and create an object of a reference type on the managed heap, use [try-finally](#) syntax to ensure that an exception doesn't prevent the destructor from running.

```
// compile with: /clr
ref struct A {
    ~A() {}
};

int main() {
    A ^ MyA = gcnew A;
    try {
        // use MyA
    }
    finally {
        delete MyA;
    }
}
```

If your type has a destructor, the compiler generates a `Dispose` method that implements [IDisposable](#). If a type that's written in Visual C++ and has a destructor that's consumed from another language, calling `IDisposable::Dispose` on that type causes the type's destructor to be called. When the type is consumed from a Visual C++ client, you can't directly call `Dispose`; instead, call the destructor by using the `delete` operator.

If your type has a finalizer, the compiler generates a `Finalize(void)` method that overrides [Finalize](#).

If a type has either a finalizer or a destructor, the compiler generates a `Dispose(bool)` method, according to the design pattern. (For information, see [Dispose Pattern](#)). You can't explicitly author or call `Dispose(bool)` in Visual C++.

If a type has a base class that conforms to the design pattern, the destructors for all base classes are called when the destructor for the derived class is called. (If your type is written in Visual C++, the compiler ensures that your types implement this pattern.) In other words, the destructor of a reference class chains to its bases and members as specified by the C++ standard. First, the class's destructor is run. Then, the destructors for its members get run in the reverse of the order in which they were constructed. Finally, the destructors for its base classes get run in the reverse of the order in which they were constructed.

Destructors and finalizers aren't allowed inside value types or interfaces.

A finalizer can only be defined or declared in a reference type. Like a constructor and destructor, a finalizer has no return type.

After an object's finalizer runs, finalizers in any base classes are also called, beginning with the least derived type. Finalizers for data members aren't automatically chained to by a class's finalizer.

If a finalizer deletes a native pointer in a managed type, you must ensure that references to or through the native pointer aren't prematurely collected. Call the destructor on the managed type instead of using [KeepAlive](#).

At compile time, you can detect whether a type has a finalizer or a destructor. For more information, see [Compiler Support for Type Traits](#).

The next sample shows two types: one that has unmanaged resources, and one that has managed resources that get released deterministically.

```

// compile with: /clr
#include <vcclr.h>
#include <stdio.h>
using namespace System;
using namespace System::IO;

ref class SystemFileWriter {
    FileStream ^ file;
    array<Byte> ^ arr;
    int bufLen;

public:
    SystemFileWriter(String ^ name) : file(File::Open(name, FileMode::Append)),
        arr(gcnew array<Byte>(1024)) {}

    void Flush() {
        file->Write(arr, 0, bufLen);
        bufLen = 0;
    }

    ~SystemFileWriter() {
        Flush();
        delete file;
    }
};

ref class CRTFileWriter {
    FILE * file;
    array<Byte> ^ arr;
    int bufLen;

    static FILE * getFile(String ^ n) {
        pin_ptr<const wchar_t> name = PtrToStringChars(n);
        FILE * ret = 0;
        _wfopen_s(&ret, name, L"ab");
        return ret;
    }

public:
    CRTFileWriter(String ^ name) : file(getFile(name)), arr(gcnew array<Byte>(1024)) {}

    void Flush() {
        pin_ptr<Byte> buf = &arr[0];
        fwrite(buf, 1, bufLen, file);
        bufLen = 0;
    }

    ~CRTFileWriter() {
        this->!CRTFileWriter();
    }

    !CRTFileWriter() {
        Flush();
        fclose(file);
    }
};

int main() {
    SystemFileWriter w("systest.txt");
    CRTFileWriter ^ w2 = gcnew CRTFileWriter("crttest.txt");
}

```

## See also

[Classes and structs](#)

# C++ Stack Semantics for Reference Types

9/20/2022 • 3 minutes to read • [Edit Online](#)

Prior to Visual Studio 2005, an instance of a reference type could only be created using the `new` operator, which created the object on the garbage collected heap. However, you can now create an instance of a reference type using the same syntax that you would use to create an instance of a native type on the stack. So, you do not need to use `ref new`, `gcnew` to create an object of a reference type. And when the object goes out of scope, the compiler calls the object's destructor.

## Remarks

When you create an instance of a reference type using stack semantics, the compiler does internally create the instance on the garbage collected heap (using `gcnew`).

When the signature or return type of a function includes an instance of a by-value reference type, the function will be marked in the metadata as requiring special handling (with modreq). This special handling is currently only provided by Visual C++ clients; other languages do not currently support consuming functions or data that use reference types created with stack semantics.

One reason to use `gcnew` (dynamic allocation) instead of stack semantics would be if the type has no destructor. Also, using reference types created with stack semantics in function signatures would not be possible if you want your functions to be consumed by languages other than Visual C++.

The compiler will not generate a copy constructor for a reference type. Therefore, if you define a function that uses a by-value reference type in the signature, you must define a copy constructor for the reference type. A copy constructor for a reference type has a signature of the following form: `R(R%){}.`

The compiler will not generate a default assignment operator for a reference type. An assignment operator allows you to create an object using stack semantics and initialize it with an existing object created using stack semantics. An assignment operator for a reference type has a signature of the following form:

`void operator=( R% ){}.`

If your type's destructor releases critical resources and you use stack semantics for reference types, you do not need to explicitly call the destructor (or call `delete`). For more information on destructors in reference types, see [Destructors and finalizers in How to: Define and consume classes and structs \(C++/CLI\)](#).

A compiler-generated assignment operator will follow the usual standard C++ rules with the following additions:

- Any non-static data members whose type is a handle to a reference type will be shallow copied (treated like a non-static data member whose type is a pointer).
- Any non-static data member whose type is a value type will be shallow copied.
- Any non-static data member whose type is an instance of a reference type will invoke a call to the reference type's copy constructor.

The compiler also provides a `%` unary operator to convert an instance of a reference type created using stack semantics to its underlying handle type.

The following reference types are not available for use with stack semantics:

- [delegate \(C++ Component Extensions\)](#)

- [Arrays](#)
- [String](#)

## Example

### Description

The following code sample shows how to declare instances of reference types with stack semantics, how the assignment operator and copy constructor works, and how to initialize a tracking reference with reference type created using stack semantics.

### Code

```
// stack_semantics_for_reference_types.cpp
// compile with: /clr
ref class R {
public:
    int i;
    R(){}
    // assignment operator
    void operator=(R% r) {
        i = r.i;
    }
    // copy constructor
    R(R% r) : i(r.i) {}
};

void Test(R r) {} // requires copy constructor

int main() {
    R r1;
    r1.i = 98;

    R r2(r1); // requires copy constructor
    System::Console::WriteLine(r1.i);
    System::Console::WriteLine(r2.i);

    // use % unary operator to convert instance using stack semantics
    // to its underlying handle
    R ^ r3 = %r1;
    System::Console::WriteLine(r3->i);

    Test(r1);

    R r4;
    R r5;
    r5.i = 13;
    r4 = r5; // requires a user-defined assignment operator
    System::Console::WriteLine(r4.i);

    // initialize tracking reference
    R % r6 = r4;
    System::Console::WriteLine(r6.i);
}
```

### Output

98

98

98

13

13

## See also

[Classes and Structs](#)

# User-Defined Operators (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

User-defined operators for managed types are allowed as static members or instance members, or at global scope. However, only static operators are accessible through metadata to clients that are written in a language other than Visual C++.

In a reference type, one of the parameters of a static user-defined operator must be one of these:

- A handle (`type ^`) to an instance of the enclosing type.
- A reference type indirection (`type ^%` or `type ^&`) to a handle to an instance of the enclosing type.

In a value type, one of the parameters of a static user-defined operator must be one of these:

- Of the same type as the enclosing value type.
- A pointer type indirection (`type ^`) to the enclosing type.
- A reference type indirection (`type %` or `type &`) to the enclosing type.
- A reference type indirection (`type ^%` or `type ^&`) to the handle.

You can define the following operators:

OPERATOR	UNARY/BINARY FORMS?
!	Unary
!=	Binary
%	Binary
&	Unary and Binary
&&	Binary
*	Unary and Binary
+	Unary and Binary
++	Unary
,	Binary
-	Unary and Binary
--	Unary
->	Unary
/	Binary

OPERATOR	UNARY/BINARY FORMS?
<	Binary
<<	Binary
<=	Binary
=	Binary
==	Binary
>	Binary
>=	Binary
>>	Binary
^	Binary
false	Unary
true	Unary
	Binary
	Binary
~	Unary

## Example: User-defined operators

```

// mcppv2_user-defined_operators.cpp
// compile with: /clr
using namespace System;
public ref struct X {
    X(int i) : m_i(i) {}
    X() {}

    int m_i;

    // static, binary, user-defined operator
    static X^ operator + (X^ me, int i) {
        return (gcnew X(me -> m_i + i));
    }

    // instance, binary, user-defined operator
    X^ operator -( int i ) {
        return gcnew X(this->m_i - i);
    }

    // instance, unary, user-defined pre-increment operator
    X^ operator ++() {
        return gcnew X(this->m_i++);
    }

    // instance, unary, user-defined post-increment operator
    X^ operator +(int i) {
        return gcnew X(this->m_i++);
    }

    // static, unary user-defined pre- and post-increment operator
    static X^ operator-- (X^ me) {
        return (gcnew X(me -> m_i - 1));
    }
};

int main() {
    X ^hX = gcnew X(-5);
    System::Console::WriteLine(hX -> m_i);

    hX = hX + 1;
    System::Console::WriteLine(hX -> m_i);

    hX = hX - (-1);
    System::Console::WriteLine(hX -> m_i);

    ++hX;
    System::Console::WriteLine(hX -> m_i);

    hX++;
    System::Console::WriteLine(hX -> m_i);

    hX--;
    System::Console::WriteLine(hX -> m_i);

    --hX;
    System::Console::WriteLine(hX -> m_i);
}

```

```

-5
-4
-3
-2
-1
-2
-3

```

## Example: Operator synthesis

The following sample demonstrates operator synthesis, which is only available when you use `/clr` to compile. Operator synthesis creates the assignment form of a binary operator, if one is not defined, where the left-hand side of the assignment operator has a CLR type.

```
// mcppv2_user-defined_operators_2.cpp
// compile with: /clr
ref struct A {
    A(int n) : m_n(n) {};
    static A^ operator + (A^ r1, A^ r2) {
        return gcnew A( r1->m_n + r2->m_n);
    };
    int m_n;
};

int main() {
    A^ a1 = gcnew A(10);
    A^ a2 = gcnew A(20);

    a1 += a2;    // a1 = a1 + a2    += not defined in source
    System::Console::WriteLine(a1->m_n);
}
```

30

## See also

[Classes and Structs](#)

# User-Defined Conversions (C++/CLI)

9/20/2022 • 4 minutes to read • [Edit Online](#)

This section discusses user-defined conversions (UDC) when one of the types in the conversion is a reference or instance of a value type or reference type.

## Implicit and explicit conversions

A user-defined conversion can either be implicit or explicit. A UDC should be implicit if the conversion does not result in a loss of information. Otherwise an explicit UDC should be defined.

A native class's constructor can be used to convert a reference or value type to a native class.

For more information about conversions, see [Boxing](#) and [Standard Conversions](#).

```
// mcpp_User Defined_Conversions.cpp
// compile with: /clr
#include "stdio.h"
ref class R;
class N;

value class V {
    static operator V(R^) {
        return V();
    }
};

ref class R {
public:
    static operator N(R^);
    static operator V(R^) {
        System::Console::WriteLine("in R::operator N");
        return V();
    }
};

class N {
public:
    N(R^) {
        printf("in N::N\n");
    }
};

R::operator N(R^) {
    System::Console::WriteLine("in R::operator N");
    return N(nullptr);
}

int main() {
    // Direct initialization:
    R ^r2;
    N n2(r2);    // direct initialization, calls constructor
    static_cast<N>(r2);    // also direct initialization

    R ^r3;
    // ambiguous V::operator V(R^) and R::operator V(R^)
    // static_cast<V>(r3);
}
```

## Output

```
in N::N  
in N::N
```

## Convert-From Operators

Convert-from operators create an object of the class in which the operator is defined from an object of some other class.

Standard C++ does not support convert-from operators; standard C++ uses constructors for this purpose. However, when using CLR types, Visual C++ provide syntactic support for calling convert-from operators.

To interoperate well with other CLS-conformant languages, you may wish to wrap each user-defined unary constructor for a given class with a corresponding convert-from operator.

Convert-from operators:

- Shall be defined as static functions.
- Can either be implicit (for conversions that do not lose precision such as short-to-int) or explicit, when there may be a loss of precision.
- Shall return an object of the containing class.
- Shall have the "from" type as the sole parameter type.

The following sample shows an implicit and explicit "convert-from", user-defined conversion (UDC) operator.

```

// clr_udc_convert_from.cpp
// compile with: /clr
value struct MyDouble {
    double d;

    MyDouble(int i) {
        d = static_cast<double>(i);
        System::Console::WriteLine("in constructor");
    }

    // Wrap the constructor with a convert-from operator.
    // implicit UDC because conversion cannot lose precision
    static operator MyDouble (int i) {
        System::Console::WriteLine("in operator");
        // call the constructor
        MyDouble d(i);
        return d;
    }

    // an explicit user-defined conversion operator
    static explicit operator signed short int (MyDouble) {
        return 1;
    }
};

int main() {
    int i = 10;
    MyDouble md = i;
    System::Console::WriteLine(md.d);

    // using explicit user-defined conversion operator requires a cast
    unsigned short int j = static_cast<unsigned short int>(md);
    System::Console::WriteLine(j);
}

```

## Output

```

in operator
in constructor
10
1

```

## Convert-to operators

Convert-to operators convert an object of the class in which the operator is defined to some other object. The following sample shows an implicit, convert-to, user-defined conversion operator:

```

// clr_udc_convert_to.cpp
// compile with: /clr
using namespace System;
value struct MyInt {
    Int32 i;

    // convert MyInt to String^
    static operator String^ ( MyInt val ) {
        return val.i.ToString();
    }

    MyInt(int _i) : i(_i) {}
};

int main() {
    MyInt mi(10);
    String ^s = mi;
    Console::WriteLine(s);
}

```

## Output

```
10
```

An explicit user-defined convert-to conversion operator is appropriate for conversions that potentially lose data in some way. To invoke an explicit convert-to operator, a cast must be used.

```

// clr_udc_convert_to_2.cpp
// compile with: /clr
value struct MyDouble {
    double d;
    // convert MyDouble to Int32
    static explicit operator System::Int32 ( MyDouble val ) {
        return (int)val.d;
    }
};

int main() {
    MyDouble d;
    d.d = 10.3;
    System::Console::WriteLine(d.d);
    int i = 0;
    i = static_cast<int>(d);
    System::Console::WriteLine(i);
}

```

## Output

```
10.3
10
```

## To convert generic classes

You can convert a generic class to T.

```

// clr_udc_generics.cpp
// compile with: /clr
generic<class T>
public value struct V {
    T mem;
    static operator T(V v) {
        return v.mem;
    }

    void f(T t) {
        mem = t;
    }
};

int main() {
    V<int> v;
    v.f(42);
    int i = v;
    i += v;
    System::Console::WriteLine(i == (42 * 2));
}

```

## Output

```
True
```

A converting constructor takes a type and uses it to create an object. A converting constructor is called with direct initialization only; casts will not invoke converting constructors. By default, converting constructors are explicit for CLR types.

```

// clr_udc_converting_constructors.cpp
// compile with: /clr
public ref struct R {
    int m;
    char c;

    R(int i) : m(i) { }
    R(char j) : c(j) { }
};

public value struct V {
    R^ ptr;
    int m;

    V(R^ r) : ptr(r) { }
    V(int i) : m(i) { }
};

int main() {
    R^ r = gcnew R(5);

    System::Console::WriteLine( V(5).m);
    System::Console::WriteLine( V(r).ptr);
}

```

## Output

```
5
R
```

In this code sample, an implicit static conversion function does the same thing as an explicit conversion constructor.

```
public value struct V {
    int m;
    V(int i) : m(i) {}
    static operator V(int i) {
        V v(i*100);
        return v;
    }
};

public ref struct R {
    int m;
    R(int i) : m(i) {}
    static operator R^(int i) {
        return gcnew R(i*100);
    }
};

int main() {
    V v(13);    // explicit
    R^ r = gcnew R(12);    // explicit

    System::Console::WriteLine(v.m);
    System::Console::WriteLine(r->m);

    // explicit ctor can't be called here: not ambiguous
    v = 5;
    r = 20;

    System::Console::WriteLine(v.m);
    System::Console::WriteLine(r->m);
}
```

## Output

```
13
12
500
2000
```

## See also

[Classes and Structs](#)

# initonly (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

**initonly** is a context-sensitive keyword that indicates that variable assignment can occur only as part of the declaration or in a static constructor in the same class.

The following example shows how to use `initonly`:

```
// mcpp_initonly.cpp
// compile with: /clr /c
ref struct Y1 {
    initonly
    static int staticConst1;

    initonly
    static int staticConst2 = 0;

    static Y1() {
        staticConst1 = 0;
    }
};
```

## See also

[Classes and Structs](#)

# How to: Define and Use Delegates (C++/CLI)

9/20/2022 • 7 minutes to read • [Edit Online](#)

This article shows how to define and consume delegates in C++/CLI.

Although the .NET Framework provides a number of delegates, sometimes you might have to define new delegates.

The following code example defines a delegate that's named `MyCallback`. The event-handling code—the function that's called when this new delegate is fired—must have a return type of `void` and take a `String` reference.

The main function uses a static method that's defined by `SomeClass` to instantiate the `MyCallback` delegate. The delegate then becomes an alternate method of calling this function, as demonstrated by sending the string "single" to the delegate object. Next, additional instances of `MyCallback` are linked together and then executed by one call to the delegate object.

```

// use_delegate.cpp
// compile with: /clr
using namespace System;

ref class SomeClass
{
public:
    static void Func(String^ str)
    {
        Console::WriteLine("static SomeClass::Func - {0}", str);
    }
};

ref class OtherClass
{
public:
    OtherClass( Int32 n )
    {
        num = n;
    }

    void Method(String^ str)
    {
        Console::WriteLine("OtherClass::Method - {0}, num = {1}",
                           str, num);
    }

    Int32 num;
};

delegate void MyCallback(String^ str);

int main( )
{
    MyCallback^ callback = gcnew MyCallback(SomeClass::Func);
    callback("single");

    callback += gcnew MyCallback(SomeClass::Func);

    OtherClass^ f = gcnew OtherClass(99);
    callback += gcnew MyCallback(f, &OtherClass::Method);

    f = gcnew OtherClass(100);
    callback += gcnew MyCallback(f, &OtherClass::Method);

    callback("chained");

    return 0;
}

```

```

static SomeClass::Func - single
static SomeClass::Func - chained
static SomeClass::Func - chained
OtherClass::Method - chained, num = 99
OtherClass::Method - chained, num = 100

```

The next code sample shows how to associate a delegate with a member of a value class.

```

// mcppv2_del_mem_value_class.cpp
// compile with: /clr
using namespace System;
public delegate void MyDel();

value class A {
public:
    void func1() {
        Console::WriteLine("test");
    }
};

int main() {
    A a;
    A^ ah = a;
    MyDel^ f = gcnew MyDel(a, &A::func1); // implicit box of a
    f();
    MyDel^ f2 = gcnew MyDel(ah, &A::func1);
    f2();
}

```

```

test
test

```

## How to compose delegates

You can use the "`-`" operator to remove a component delegate from a composed delegate.

```

// mcppv2_compose_delegates.cpp
// compile with: /clr
using namespace System;

delegate void MyDelegate(String ^ s);

ref class MyClass {
public:
    static void Hello(String ^ s) {
        Console::WriteLine("Hello, {0}!", s);
    }

    static void Goodbye(String ^ s) {
        Console::WriteLine(" Goodbye, {0}!", s);
    }
};

int main() {

    MyDelegate ^ a = gcnew MyDelegate(MyClass::Hello);
    MyDelegate ^ b = gcnew MyDelegate(MyClass::Goodbye);
    MyDelegate ^ c = a + b;
    MyDelegate ^ d = c - a;

    Console::WriteLine("Invoking delegate a:");
    a("A");
    Console::WriteLine("Invoking delegate b:");
    b("B");
    Console::WriteLine("Invoking delegate c:");
    c("C");
    Console::WriteLine("Invoking delegate d:");
    d("D");
}

```

## Output

```
Invoking delegate a:  
Hello, A!  
Invoking delegate b:  
Goodbye, B!  
Invoking delegate c:  
Hello, C!  
Goodbye, C!  
Invoking delegate d:  
Goodbye, D!
```

## Pass a delegate<sup>^</sup> to a native function that expects a function pointer

From a managed component you can call a native function with function pointer parameters where the native function then can call the member function of the managed component's delegate.

This sample creates the .dll that exports the native function:

```
// delegate_to_native_function.cpp  
// compile with: /LD  
#include < windows.h >  
extern "C" {  
    __declspec(dllexport)  
    void nativeFunction(void (CALLBACK *mgdFunc)(const char* str)) {  
        mgdFunc("Call to Managed Function");  
    }  
}
```

The next sample consumes the .dll and passes a delegate handle to the native function that expects a function pointer.

```
// delegate_to_native_function_2.cpp  
// compile with: /clr  
using namespace System;  
using namespace System::Runtime::InteropServices;  
  
delegate void Del(String ^s);  
public ref class A {  
public:  
    void delMember(String ^s) {  
        Console::WriteLine(s);  
    }  
};  
  
[DllImportAttribute("delegate_to_native_function", CharSet=CharSet::Ansi)]  
extern "C" void nativeFunction(Del ^d);  
  
int main() {  
    A ^a = gcnew A;  
    Del ^d = gcnew Del(a, &A::delMember);  
    nativeFunction(d); // Call to native function  
}
```

## Output

```
Call to Managed Function
```

## To associate delegates with unmanaged functions

To associate a delegate with a native function, you must wrap the native function in a managed type and declare the function to be invoked through `PInvoke`.

```
// mcppv2_del_to_umnangd_func.cpp
// compile with: /clr
#pragma unmanaged
extern "C" void printf(const char*, ...);
class A {
public:
    static void func(char* s) {
        printf(s);
    }
};

#pragma managed
public delegate void func(char*);

ref class B {
    A* ap;

public:
    B(A* ap):ap(ap) {}
    void func(char* s) {
        ap->func(s);
    }
};

int main() {
    A* a = new A;
    B^ b = gcnew B(a);
    func^ f = gcnew func(b, &B::func);
    f("hello");
    delete a;
}
```

### Output

```
hello
```

## To use unbound delegates

You can use an unbound delegate to pass an instance of the type whose function you want to call when the delegate is called.

Unbound delegates are especially useful if you want to iterate through the objects in a collection—by using `for each, in` keywords—and call a member function on each instance.

Here's how to declare, instantiate, and call bound and unbound delegates:

ACTION	BOUND DELEGATES	UNBOUND DELEGATES
--------	-----------------	-------------------

ACTION	BOUND DELEGATES	UNBOUND DELEGATES
Declare	<p>The delegate signature must match the signature of the function you want to call through the delegate.</p>	<p>The first parameter of the delegate signature is the type of <code>this</code> for the object you want to call.</p> <p>After the first parameter, the delegate signature must match the signature of the function you want to call through the delegate.</p>
Instantiate	<p>When you instantiate a bound delegate, you can specify an instance function, or a global or static member function.</p> <p>To specify an instance function, the first parameter is an instance of the type whose member function you want to call and the second parameter is the address of the function you want to call.</p> <p>If you want to call a global or static member function, just pass the name of a global function or the name of the static member function.</p>	<p>When you instantiate an unbound delegate, just pass the address of the function you want to call.</p>
Call	<p>When you call a bound delegate, just pass the parameters that are required by the delegate signature.</p>	<p>Same as a bound delegate, but remember that the first parameter must be an instance of the object that contains the function you want to call.</p>

This sample demonstrates how to declare, instantiate, and call unbound delegates:

```

// unbound_delegates.cpp
// compile with: /clr
ref struct A {
    A(){}
    A(int i) : m_i(i){}
    void Print(int i) { System::Console::WriteLine(m_i + i);}

private:
    int m_i;
};

value struct V {
    void Print() { System::Console::WriteLine(m_i);}
    int m_i;
};

delegate void Delegate1(A^, int i);
delegate void Delegate2(A%, int i);

delegate void Delegate3(interior_ptr<V>);
delegate void Delegate4(V%);

delegate void Delegate5(int i);
delegate void Delegate6();

int main() {
    A^ a1 = gcnew A(1);
    A% a2 = *gcnew A(2);

    Delegate1 ^ Unbound_Delegate1 = gcnew Delegate1(&A::Print);
    // delegate takes a handle
    Unbound_Delegate1(a1, 1);
    Unbound_Delegate1(%a2, 1);

    Delegate2 ^ Unbound_Delegate2 = gcnew Delegate2(&A::Print);
    // delegate takes a tracking reference (must deference the handle)
    Unbound_Delegate2(*a1, 1);
    Unbound_Delegate2(a2, 1);

    // instantiate a bound delegate to an instance member function
    Delegate5 ^ Bound_Del = gcnew Delegate5(a1, &A::Print);
    Bound_Del(1);

    // instantiate value types
    V v1 = {7};
    V v2 = {8};

    Delegate3 ^ Unbound_Delegate3 = gcnew Delegate3(&V::Print);
    Unbound_Delegate3(&v1);
    Unbound_Delegate3(&v2);

    Delegate4 ^ Unbound_Delegate4 = gcnew Delegate4(&V::Print);
    Unbound_Delegate4(v1);
    Unbound_Delegate4(v2);

    Delegate6 ^ Bound_Delegate3 = gcnew Delegate6(v1, &V::Print);
    Bound_Delegate3();
}

```

## Output

```
2
3
2
3
2
7
8
7
8
7
```

The next sample shows how to use unbound delegates and the `for each, in` keywords to iterate through objects in a collection and call a member function on each instance.

```
// unbound_delegates_2.cpp
// compile with: /clr
using namespace System;

ref class RefClass {
    String^ _Str;

public:
    RefClass( String^ str ) : _Str( str ) {}
    void Print() { Console::Write( _Str ); }
};

delegate void PrintDelegate( RefClass^ );

int main() {
    PrintDelegate^ d = gcnew PrintDelegate( &RefClass::Print );

    array< RefClass^ >^ a = gcnew array<RefClass^>( 10 );

    for ( int i = 0; i < a->Length; ++i )
        a[i] = gcnew RefClass( i.ToString() );

    for each ( RefClass^ R in a )
        d( R );

    Console::WriteLine();
}
```

This sample creates an unbound delegate to a property's accessor functions:

```

// unbound_delegates_3.cpp
// compile with: /clr
ref struct B {
    property int P1 {
        int get() { return m_i; }
        void set(int i) { m_i = i; }
    }

private:
    int m_i;
};

delegate void DelBSet(B^, int);
delegate int DelBGet(B^);

int main() {
    B^ b = gcnew B;

    DelBSet^ delBSet = gcnew DelBSet(&B::P1::set);
    delBSet(b, 11);

    DelBGet^ delBGet = gcnew DelBGet(&B::P1::get);
    System::Console::WriteLine(delBGet(b));
}

```

## Output

```
11
```

The following sample shows how to invoke a multicast delegate, where one instance is bound and one instance is unbound.

```

// unbound_delegates_4.cpp
// compile with: /clr
ref class R {
public:
    R(int i) : m_i(i) {}

    void f(R ^ r) {
        System::Console::WriteLine("in f(R ^ r)");
    }

    void f() {
        System::Console::WriteLine("in f()");
    }

private:
    int m_i;
};

delegate void Del(R ^);

int main() {
    R ^r1 = gcnew R(11);
    R ^r2 = gcnew R(12);

    Del^ d = gcnew Del(r1, &R::f);
    d += gcnew Del(&R::f);
    d(r2);
}

```

## Output

```
in f(R ^ r)
in f()
```

The next sample shows how to create and call an unbound generic delegate.

```
// unbound_delegates_5.cpp
// compile with: /clr
ref struct R {
    R(int i) : m_i(i) {}

    int f(R ^) { return 999; }
    int f() { return m_i + 5; }

    int m_i;
};

value struct V {
    int f(V%) { return 999; }
    int f() { return m_i + 5; }

    int m_i;
};

generic <typename T>
delegate int Del(T t);

generic <typename T>
delegate int DelV(T% t);

int main() {
    R^ hr = gcnew R(7);
    System::Console::WriteLine((gcnew Del<R^>(&R::f))(hr));

    V v;
    v.m_i = 9;
    System::Console::WriteLine((gcnew DelV<V >(&V::f))(v));
}
```

## Output

```
12
14
```

## See also

[delegate \(C++ Component Extensions\)](#)

# How to: Define and consume enums in C++/CLI

9/20/2022 • 2 minutes to read • [Edit Online](#)

Enumeration types in C++/CLI have some differences with enumeration types in standard C++. This article explains how to use C++/CLI enumeration types and how to interoperate with standard enumeration types.

## Specifying the underlying type of an `enum`

By default, the underlying type of an enumeration is `int`. However, you can specify the type to be signed or unsigned forms of `int`, `short`, `long`, `_int32`, or `_int64`. You can also use `char`.

```
// mcppv2_enum_3.cpp
// compile with: /clr
public enum class day_char : char {sun, mon, tue, wed, thu, fri, sat};

int main() {
    // fully qualified names, enumerator not injected into scope
    day_char d = day_char::sun, e = day_char::mon;
    System::Console::WriteLine(d);
    char f = (char)d;
    System::Console::WriteLine(f);
    f = (char)e;
    System::Console::WriteLine(f);
    e = day_char::tue;
    f = (char)e;
    System::Console::WriteLine(f);
}
```

## Output

```
sun
0
1
2
```

## How to convert between managed and standard enumerations

There's no standard conversion between an enum and an integral type; a cast is required.

```
// mcppv2_enum_4.cpp
// compile with: /clr
enum class day {sun, mon, tue, wed, thu, fri, sat};
enum {sun, mon, tue, wed, thu, fri, sat} day2; // unnamed std enum

int main() {
    day a = day::sun;
    day2 = sun;
    if ((int)a == day2)
        // or...
        // if (a == (day)day2)
        System::Console::WriteLine("a and day2 are the same");
    else
        System::Console::WriteLine("a and day2 are not the same");
}
```

## Output

```
a and day2 are the same
```

## Operators and enums

The following operators are valid on enums in C++/CLI:

OPERATOR
<code>==</code> <code>!=</code> <code>&lt;</code> <code>&gt;</code> <code>&lt;=</code> <code>&gt;=</code>
<code>+</code> <code>-</code>
<code> </code> <code>^</code> <code>&amp;</code> <code>~</code>
<code>++</code> <code>--</code>
<code>sizeof</code>

Operators `|`, `^`, `&`, `~`, `++`, and `--` are defined only for enumerations with integral underlying types, not including `bool`. Both operands must be of the enumeration type.

The compiler does no static or dynamic checking of the result of an enum operation; an operation may result in a value not in the range of the enum's valid enumerators.

### NOTE

C++11 introduces `enum class` types in unmanaged code, which are significantly different than managed `enum class` types in C++/CLI. In particular, the C++11 `enum class` type does not support the same operators as the managed `enum class` type in C++/CLI, and C++/CLI source code must provide an accessibility specifier in managed `enum class` declarations in order to distinguish them from unmanaged (C++11) `enum class` declarations. For more information about `enum class` use in C++/CLI, C++/CX, and C++11, see [enum class](#).

```
// mcppv2_enum_5.cpp
// compile with: /clr
private enum class E { a, b } e, mask;
int main() {
    if ( e & mask )    // C2451 no E->bool conversion
        ;

    if ( ( e & mask ) != 0 )    // C3063 no operator!= (E, int)
        ;

    if ( ( e & mask ) != E() )    // OK
        ;
}
```

Use scope qualifiers to distinguish between `enum` and `enum class` values:

```
// mcppv2_enum_6.cpp
// compile with: /clr
private enum class day : int {sun, mon};
enum : bool {sun = true, mon = false} day2;

int main() {
    day a = day::sun, b = day::mon;
    day2 = sun;

    System::Console::WriteLine(sizeof(a));
    System::Console::WriteLine(sizeof(day2));
    a++;
    System::Console::WriteLine(a == b);
}
```

## Output

```
4
1
True
```

## See also

[enum class](#)

# How to: Use Events in C++/CLI

9/20/2022 • 8 minutes to read • [Edit Online](#)

This article shows how to use an interface that declares an event and a function to invoke that event, and the class and event handler that implement the interface.

## Interface events

The following code example adds an event handler, invokes the event—which causes the event handler to write its name to the console—and then removes the event handler.

```
// mcppv2_events2.cpp
// compile with: /clr
using namespace System;

delegate void Del(int, float);

// interface that has an event and a function to invoke the event
interface struct I {
public:
    event Del ^ E;
    void fire(int, float);
};

// class that implements the interface event and function
ref class EventSource: public I {
public:
    virtual event Del^ E;
    virtual void fire(int i, float f) {
        E(i, f);
    }
};

// class that defines the event handler
ref class EventReceiver {
public:
    void Handler(int i , float f) {
        Console::WriteLine("EventReceiver::Handler");
    }
};

int main () {
    I^ es = gcnew EventSource();
    EventReceiver^ er = gcnew EventReceiver();

    // hook the handler to the event
    es->E += gcnew Del(er, &EventReceiver::Handler);

    // call the event
    es -> fire(1, 3.14);

    // unhook the handler from the event
    es->E -= gcnew Del(er, &EventReceiver::Handler);
}
```

## Output

```
EventReceiver::Handler
```

## Custom accessor methods

The following sample shows how to define an event's behavior when handlers are added or removed, and when an event is raised.

```
// mcppv2_events6.cpp
// compile with: /clr
using namespace System;

public delegate void MyDel();
public delegate int MyDel2(int, float);

ref class EventSource {
public:
    MyDel ^ pE;
    MyDel2 ^ pE2;

    event MyDel^ E {
        void add(MyDel^ p) {
            pE = static_cast<MyDel^> (Delegate::Combine(pE, p));
            // cannot refer directly to the event
            // E = static_cast<MyDel^> (Delegate::Combine(pE, p));    // error
        }
        void remove(MyDel^ p) {
            pE = static_cast<MyDel^> (Delegate::Remove(pE, p));
        }
        void raise() {
            if (pE != nullptr)
                pE->Invoke();
        }
    } // E event block

    event MyDel2^ E2 {
        void add(MyDel2^ p2) {
            pE2 = static_cast<MyDel2^> (Delegate::Combine(pE2, p2));
        }
        void remove(MyDel2^ p2) {
            pE2 = static_cast<MyDel2^> (Delegate::Remove(pE2, p2));
        }
        int raise(int i, float f) {
            if (pE2 != nullptr) {
                return pE2->Invoke(i, f);
            }
            return 1;
        }
    } // E2 event block
};

public ref struct EventReceiver {
    void H1() {
        Console::WriteLine("In event handler H1");
    }

    int H2(int i, float f) {
        Console::WriteLine("In event handler H2 with args {0} and {1}", i.ToString(), f.ToString());
        return 0;
    }
};
```

```

int main() {
    EventSource ^ pE = gcnew EventSource;
    EventReceiver ^ pR = gcnew EventReceiver;

    // hook event handlers
    pE->E += gcnew MyDel(pR, &EventReceiver::H1);
    pE->E2 += gcnew MyDel2(pR, &EventReceiver::H2);

    // raise events
    pE->E();
    pE->E2::raise(1, 2.2);    // call event through scope path

    // unhook event handlers
    pE->E -= gcnew MyDel(pR, &EventReceiver::H1);
    pE->E2 -= gcnew MyDel2(pR, &EventReceiver::H2);

    // raise events, but no handlers
    pE->E();
    pE->E2::raise(1, 2.5);
}

```

## Output

```

In event handler H1
In event handler H2 with args 1 and 2.2

```

## Override default access on add, remove, and raise accessors

This sample shows how to override the default access on the add, remove, and raise events methods:

```

// mcppv2_events3.cpp
// compile with: /clr
public delegate void f(int);

public ref struct E {
    f ^ _E;
public:
    void handler(int i) {
        System::Console::WriteLine(i);
    }

    E() {
        _E = nullptr;
    }

    event f^ Event {
        void add(f ^ d) {
            _E += d;
        }
    private:
        void remove(f ^ d) {
            _E -= d;
        }
    }

    protected:
        void raise(int i) {
            if (_E) {
                _E->Invoke(i);
            }
        }
    }
}

// a member function to access all event methods
static void Go() {
    E^ pE = gcnew E;
    pE->Event += gcnew f(pE, &E::handler);
    pE->Event(17); // prints 17
    pE->Event -= gcnew f(pE, &E::handler);
    pE->Event(17); // no output
}
};

int main() {
    E::Go();
}

```

## Output

17

## Multiple event handlers

An event receiver, or any other client code, can add one or more handlers to an event.

```

// mcppv2_events4.cpp
// compile with: /clr
using namespace System;
#include <stdio.h>

delegate void ClickEventHandler(int, double);
delegate void DblClickEventHandler(String^);

ref class EventSource {
public:
    event ClickEventHandler^ OnClick;
    event DblClickEventHandler^ OnDblClick;

    void FireEvents() {
        OnClick(7, 3.14159);
        OnDblClick("Started");
    }
};

ref struct EventReceiver {
public:
    void Handler1(int x, double y) {
        System::Console::Write("Click(x={0},y={1})\n", x, y);
    }

    void Handler2(String^ s) {
        System::Console::Write("DblClick(s={0})\n", s);
    }

    void Handler3(String^ s) {
        System::Console::WriteLine("DblClickAgain(s={0})\n", s);
    }

    void AddHandlers(EventSource^ pES) {
        pES->OnClick +=
            gcnew ClickEventHandler(this,&EventReceiver::Handler1);
        pES->OnDblClick +=
            gcnew DblClickEventHandler(this,&EventReceiver::Handler2);
        pES->OnDblClick +=
            gcnew DblClickEventHandler(this, &EventReceiver::Handler3);
    }

    void RemoveHandlers(EventSource^ pES) {
        pES->OnClick -=
            gcnew ClickEventHandler(this, &EventReceiver::Handler1);
        pES->OnDblClick -=
            gcnew DblClickEventHandler(this, &EventReceiver::Handler2);
        pES->OnDblClick -=
            gcnew DblClickEventHandler(this, &EventReceiver::Handler3);
    }
};

int main() {
    EventSource^ pES = gcnew EventSource;
    EventReceiver^ pER = gcnew EventReceiver;

    // add handlers
    pER->AddHandlers(pES);

    pES->FireEvents();

    // remove handlers
    pER->RemoveHandlers(pES);
}

```

## Output

```
Click(x=7,y=3.14159)
DblClick(s=System.Char[])
DblClickAgain(s=System.Char[])
```

## Static events

The following sample shows how to define and use static events.

```
// mcppv2_events7.cpp
// compile with: /clr
using namespace System;

public delegate void MyDel();
public delegate int MyDel2(int, float);

ref class EventSource {
public:
    static MyDel ^ psE;
    static event MyDel2 ^ E2; // event keyword, compiler generates add,
                           // remove, and Invoke

    static event MyDel ^ E {
        static void add(MyDel ^ p) {
            psE = static_cast<MyDel^> (Delegate::Combine(psE, p));
        }
        static void remove(MyDel^ p) {
            psE = static_cast<MyDel^> (Delegate::Remove(psE, p));
        }
        static void raise() {
            if (psE != nullptr) //psE!=0 -> C2679, use nullptr
                psE->Invoke();
        }
    }

    static int Fire_E2(int i, float f) {
        return E2(i, f);
    }
};

public ref struct EventReceiver {
    void H1() {
        Console::WriteLine("In event handler H1");
    }

    int H2(int i, float f) {
        Console::WriteLine("In event handler H2 with args {0} and {1}", i.ToString(), f.ToString());
        return 0;
    }
};

int main() {
    EventSource^ pE = gcnew EventSource;
    EventReceiver^ pR = gcnew EventReceiver;

    // Called with "this"
    // hook event handlers
    pE->E += gcnew MyDel(pR, &EventReceiver::H1);
    pE->E2 += gcnew MyDel2(pR, &EventReceiver::H2);

    // raise events
    pE->E();
    pE->Fire_E2(11, 11.11);
```

```

// unhook event handlers
pE->E -= gcnew MyDel(pR, &EventReceiver::H1);
pE->E2 -= gcnew MyDel2(pR, &EventReceiver::H2);

// Not called with "this"
// hook event handler
EventSource::E += gcnew MyDel(pR, &EventReceiver::H1);
EventSource::E2 += gcnew MyDel2(pR, &EventReceiver::H2);

// raise events
EventSource::E();
EventSource::Fire_E2(22, 22.22);

// unhook event handlers
EventSource::E -= gcnew MyDel(pR, &EventReceiver::H1);
EventSource::E2 -= gcnew MyDel2(pR, &EventReceiver::H2);
}

```

## Output

```

In event handler H1
In event handler H2 with args 11 and 11.11
In event handler H1
In event handler H2 with args 22 and 22.22

```

## Virtual events

This sample implements virtual, managed events in an interface and class:

```

// mcppv2_events5.cpp
// compile with: /clr
using namespace System;

public delegate void MyDel();
public delegate int MyDel2(int, float);

// managed class that has a virtual event
ref class IEFace {
public:
    virtual event MyDel ^ E; // declares three accessors (add, remove, and raise)
};

// managed interface that has a virtual event
public interface struct IEFace2 {
public:
    event MyDel2 ^ E2; // declares two accessors (add and remove)
};

// implement virtual events
ref class EventSource : public IEFace, public IEFace2 {
public:
    virtual event MyDel2 ^ E2;

    void Fire_E() {
        E();
    }

    int Fire_E2(int i, float f) {
        try {
            return E2(i, f);
        }
        catch(System::NullReferenceException^) {
            return 0; // no handlers
        }
    }
}

```

```

    }

// class to hold event handlers, the event receiver
public ref struct EventReceiver {
    // first handler
    void H1() {
        Console::WriteLine("In handler H1");
    }

    // second handler
    int H2(int i, float f) {
        Console::WriteLine("In handler H2 with args {0} and {1}", i.ToString(), f.ToString());
        return 0;
    }
};

int main() {
    EventSource ^ pE = gcnew EventSource;
    EventReceiver ^ pR = gcnew EventReceiver;

    // add event handlers
    pE->E += gcnew MyDel(pR, &EventReceiver::H1);
    pE->E2 += gcnew MyDel2(pR, &EventReceiver::H2);

    // raise events
    pE->Fire_E();
    pE->Fire_E2(1, 2.2);

    // remove event handlers
    pE->E -= gcnew MyDel(pR, &EventReceiver::H1);
    pE->E2 -= gcnew MyDel2(pR, &EventReceiver::H2);

    // raise events, but no handlers; so, no effect
    pE->Fire_E();
    pE->Fire_E2(1, 2.5);
}

```

## Output

```

In handler H1
In handler H2 with args 1 and 2.2

```

A simple event cannot be specified to override or hide a base class event. You must define all of the event's accessor functions, and then specify the `new` or `override` keyword on each accessor function.

```

// mcppv2_events5_a.cpp
// compile with: /clr /c
delegate void Del();

ref struct A {
    virtual event Del ^E;
    virtual event Del ^E2;
};

ref struct B : A {
    virtual event Del ^E override; // C3797
    virtual event Del ^E2 new; // C3797
};

ref struct C : B {
    virtual event Del ^E { // OK
        void raise() override {}
        void add(Del ^) override {}
        void remove(Del^) override {}
    }

    virtual event Del ^E2 { // OK
        void raise() new {}
        void add(Del ^) new {}
        void remove(Del^) new {}
    }
};

```

## Abstract events

The following sample shows how to implement an abstract event.

```

// mcppv2_events10.cpp
// compile with: /clr /W1
using namespace System;
public delegate void Del();
public delegate void Del2(String^ s);

interface struct IEvent {
public:
    // in this case, no raised method is defined
    event Del^ Event1;

    event Del2^ Event2 {
public:
        void add(Del2^ _d);
        void remove(Del2^ _d);
        void raise(String^ s);
    }

    void fire();
};

ref class EventSource: public IEvent {
public:
    virtual event Del^ Event1;
    event Del2^ Event2 {
        virtual void add(Del2^ _d) {
            d = safe_cast<Del2^>(System::Delegate::Combine(d, _d));
        }
    }

    virtual void remove(Del2^ _d) {
        d = safe_cast<Del2^>(System::Delegate::Remove(d, _d));
    }
};

```

```

        virtual void raise(String^ s) {
            if (d) {
                d->Invoke(s);
            }
        }

        virtual void fire() {
            return Event1();
        }

private:
    Del2^ d;
};

ref class EventReceiver {
public:
    void func() {
        Console::WriteLine("hi");
    }

    void func(String^ str) {
        Console::WriteLine(str);
    }
};

int main () {
    IEvent^ es = gcnew EventSource;
    EventReceiver^ er = gcnew EventReceiver;
    es->Event1 += gcnew Del(er, &EventReceiver::func);
    es->Event2 += gcnew Del2(er, &EventReceiver::func);

    es->fire();
    es->Event2("hello from Event2");
    es->Event1 -= gcnew Del(er, &EventReceiver::func);
    es->Event2 -= gcnew Del2(er, &EventReceiver::func);
    es->Event2("hello from Event2");
}

```

## Output

```

hi
hello from Event2

```

## Raising events that are defined in a different assembly

An event and event handler can be defined in one assembly, and consumed by another assembly.

```

// mcppv2_events8.cpp
// compile with: /LD /clr
using namespace System;

public delegate void Del(String^ s);

public ref class Source {
public:
    event Del^ Event;
    void Fire(String^ s) {
        Event(s);
    }
};

```

This client code consumes the event:

```
// mcppv2_events9.cpp
// compile with: /clr
#using "mcppv2_events8.dll"
using namespace System;

ref class Receiver {
public:
    void Handler(String^ s) {
        Console::WriteLine(s);
    }
};

int main() {
    Source^ src = gcnew Source;
    Receiver^ rc1 = gcnew Receiver;
    Receiver^ rc2 = gcnew Receiver;
    src -> Event += gcnew Del(rc1, &Receiver::Handler);
    src -> Event += gcnew Del(rc2, &Receiver::Handler);
    src->Fire("hello");
    src -> Event -= gcnew Del(rc1, &Receiver::Handler);
    src -> Event -= gcnew Del(rc2, &Receiver::Handler);
}
```

## Output

```
hello
hello
```

## See also

[event](#)

# How to: Define an Interface Static Constructor (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

An interface can have a static constructor, which can be used to initialize static data members. A static constructor will be called at most once, and will be called before the first time a static interface member is accessed.

## Example

```
// mcppv2_interface_class2.cpp
// compile with: /clr
using namespace System;

interface struct MyInterface {
    static int i;
    static void Test() {
        Console::WriteLine(i);
    }

    static MyInterface() {
        Console::WriteLine("in MyInterface static constructor");
        i = 99;
    }
};

ref class MyClass : public MyInterface {};

int main() {
    MyInterface::Test();
    MyClass::MyInterface::Test();

    MyInterface ^ mi = gcnew MyClass;
    mi->Test();
}
```

```
in MyInterface static constructor
99
99
99
```

## See also

[interface class](#)

# How to: Declare Override Specifiers in Native Compilations (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

`sealed`, `abstract`, and `override` are available in compilations that do not use `/ZW` or `/clr`.

## NOTE

The ISO C++11 Standard language has the `override` identifier and the `final` identifier, and both are supported in Visual Studio. Use `final` instead of `sealed` in code that is meant to be compiled as native-only.

## Example: sealed is valid

### Description

The following example shows that `sealed` is valid in native compilations.

### Code

```
// sealed_native_keyword.cpp
#include <stdio.h>
__interface I1 {
    virtual void f();
    virtual void g();
};

class X : public I1 {
public:
    virtual void g() sealed {};
};

class Y : public X {
public:

    // the following override generates a compiler error
    virtual void g() {} // C3248 X::g is sealed!
};
```

## Example: override is valid

### Description

The next example shows that `override` is valid in native compilations.

### Code

```
// override_native_keyword.cpp
#include <stdio.h>
__interface I1 {
    virtual void f();
};

class X : public I1 {
public:
    virtual void f() override {} // OK
    virtual void g() override {} // C3668 I1::g does not exist
};
```

## Example: abstract is valid

### Description

This example shows that `abstract` is valid in native compilations.

### Code

```
// abstract_native_keyword.cpp
class X abstract {};

int main() {
    X * MyX = new X; // C3622 cannot instantiate abstract class
}
```

## See also

[Override Specifiers](#)

# How to: Use Properties in C++/CLI

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article shows how to use properties in C++/CLI.

## Basic properties

For basic properties—those that merely assign and retrieve a private data member—you don't have to explicitly define the get and set accessor functions because the compiler automatically provides them when given just the data type of the property. This code demonstrates a basic property:

```
// SimpleProperties.cpp
// compile with: /clr
using namespace System;

ref class C {
public:
    property int Size;
};

int main() {
    C^ c = gcnew C;
    c->Size = 111;
    Console::WriteLine("c->Size = {0}", c->Size);
}
```

```
c->Size = 111
```

## Static properties

This code sample shows how to declare and use a static property. A static property can only access static members of its class.

```
// mcppv2_property_3.cpp
// compile with: /clr
using namespace System;

ref class StaticProperties {
    static int MyInt;
    static int MyInt2;

public:
    static property int Static_Data_Member_Property;

    static property int Static_Block_Property {
        int get() {
            return MyInt;
        }

        void set(int value) {
            MyInt = value;
        }
    }
};

int main() {
    StaticProperties::Static_Data_Member_Property = 96;
    Console::WriteLine(StaticProperties::Static_Data_Member_Property);

    StaticProperties::Static_Block_Property = 47;
    Console::WriteLine(StaticProperties::Static_Block_Property);
}
```

```
96
47
```

## Indexed properties

An indexed property typically exposes a data structure that's accessed by using a subscript operator.

If you use a default indexed property, you can access the data structure just by referring to the class name, but if you use a user-defined indexed property, you must specify the property name to access the data structure.

For information about how to consume an indexer that's written in C#, see [How to: Consume a C# Indexer \(C++/CLI\)](#).

This code sample shows how to use default and user-defined indexed properties:

```

// mcppv2_property_2.cpp
// compile with: /clr
using namespace System;
public ref class C {
    array<int>^ MyArr;

public:
    C() {
        MyArr = gcnew array<int>(5);
    }

    // default indexer
    property int default[int] {
        int get(int index) {
            return MyArr[index];
        }
        void set(int index, int value) {
            MyArr[index] = value;
        }
    }

    // user-defined indexer
    property int indexer1[int] {
        int get(int index) {
            return MyArr[index];
        }
        void set(int index, int value) {
            MyArr[index] = value;
        }
    }
};

int main() {
    C ^ MyC = gcnew C();

    // use the default indexer
    Console::Write("[ ");
    for (int i = 0 ; i < 5 ; i++) {
        MyC[i] = i;
        Console::Write("{0} ", MyC[i]);
    }

    Console::WriteLine("]");

    // use the user-defined indexer
    Console::Write("[ ");
    for (int i = 0 ; i < 5 ; i++) {
        MyC->indexer1[i] = i * 2;
        Console::Write("{0} ", MyC->indexer1[i]);
    }

    Console::WriteLine("]");
}

```

```
[ 0 1 2 3 4 ]
[ 0 2 4 6 8 ]
```

The next sample shows how to call the default indexer by using the `this` pointer.

```

// call_default_indexer_through_this_pointer.cpp
// compile with: /clr /c
value class Position {
public:
    Position(int x, int y) : position(gcnew array<int, 2>(100, 100)) {
        this->default[x, y] = 1;
    }

    property int default[int, int] {
        int get(int x, int y) {
            return position[x, y];
        }

        void set(int x, int y, int value) {}
    }

private:
    array<int, 2> ^ position;
};

```

This sample shows how to use [DefaultMemberAttribute](#) to specify the default indexer:

```

// specify_default_indexer.cpp
// compile with: /LD /clr
using namespace System;
[Reflection::DefaultMember("XXX")]
public ref struct Squares {
    property Double XXX[Double] {
        Double get(Double data) {
            return data*data;
        }
    }
};

```

The next sample consumes the metadata that's created in the previous example.

```

// consume_default_indexer.cpp
// compile with: /clr
#using "specify_default_indexer.dll"
int main() {
    Squares ^ square = gcnew Squares();
    System::Console::WriteLine("{0}", square[3]);
}

```

9

## Virtual properties

This code sample shows how to declare and use virtual properties:

```

// mcppv2_property_4.cpp
// compile with: /clr
using namespace System;
interface struct IEFace {
public:
    property int VirtualProperty1;
    property int VirtualProperty2 {
        int get();
        void set(int i);
    }
};

// implement virtual events
ref class PropImpl : public IEFace {
    int MyInt;
public:
    virtual property int VirtualProperty1;

    virtual property int VirtualProperty2 {
        int get() {
            return MyInt;
        }
        void set(int i) {
            MyInt = i;
        }
    }
};

int main() {
    PropImpl ^ MyPI = gcnew PropImpl();
    MyPI->VirtualProperty1 = 93;
    Console::WriteLine(MyPI->VirtualProperty1);

    MyPI->VirtualProperty2 = 43;
    Console::WriteLine(MyPI->VirtualProperty2);
}

```

93  
43

## Abstract and sealed properties

Although the `abstract` and `sealed` keywords are specified as valid in the ECMA C++/CLI specification, for the Microsoft C++ compiler, you cannot specify them on trivial properties, nor on the property declaration of a non-trivial property.

To declare a sealed or abstract property, you must define a non-trivial property and then specify the `abstract` or `sealed` keyword on the get and set accessor functions.

```

// properties_abstract_sealed.cpp
// compile with: /clr
ref struct A {
protected:
    int m_i;

public:
    A() { m_i = 87; }

    // define abstract property
    property int Prop_1 {
        virtual int get() abstract;
        virtual void set(int i) abstract;
    }
};

ref struct B : A {
private:
    int m_i;

public:
    B() { m_i = 86; }

    // implement abstract property
    property int Prop_1 {
        virtual int get() override { return m_i; }
        virtual void set(int i) override { m_i = i; }
    }
};

ref struct C {
private:
    int m_i;

public:
    C() { m_i = 87; }

    // define sealed property
    property int Prop_2 {
        virtual int get() sealed { return m_i; }
        virtual void set(int i) sealed { m_i = i; };
    }
};

int main() {
    B b1;
    // call implementation of abstract property
    System::Console::WriteLine(b1.Prop_1);

    C c1;
    // call sealed property
    System::Console::WriteLine(c1.Prop_2);
}

```

86  
87

## Multidimensional properties

You can use multidimensional properties to define property accessor methods that take a non-standard number of parameters.

```

// mcppv2_property_5.cpp
// compile with: /clr
ref class X {
    double d;
public:
    X() : d(0) {}
    property double MultiDimProp[int, int, int] {
        double get(int, int, int) {
            return d;
        }
        void set(int i, int j, int k, double l) {
            // do something with those ints
            d = l;
        }
    }

    property double MultiDimProp2[int] {
        double get(int) {
            return d;
        }
        void set(int i, double l) {
            // do something with those ints
            d = l;
        }
    }
};

int main() {
    X ^ MyX = gcnew X();
    MyX->MultiDimProp[0,0,0] = 1.1;
    System::Console::WriteLine(MyX->MultiDimProp[0, 0, 0]);
}

```

1.1

## Overloading property accessors

The following example shows how to overload indexed properties.

```
// mcppv2_property_6.cpp
// compile with: /clr
ref class X {
    double d;
public:
    X() : d(0.0) {}
    property double MyProp[int] {
        double get(int i) {
            return d;
        }
        double get(System::String ^ i) {
            return 2*d;
        }
        void set(int i, double l) {
            d = i * l;
        }
    } // end MyProp definition
};

int main() {
    X ^ MyX = gcnew X();
    MyX->MyProp[2] = 1.7;
    System::Console::WriteLine(MyX->MyProp[1]);
    System::Console::WriteLine(MyX->MyProp["test"]);
}
```

3.4  
6.8

## See also

[property](#)

# How to: Use safe\_cast in C++/CLI

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article shows how to use `safe_cast` in C++/CLI applications. For information about `safe_cast` in C++/CX, see [safe\\_cast](#).

## Upcasting

An upcast is a cast from a derived type to one of its base classes. This cast is safe and does not require an explicit cast notation. The following sample shows how to perform an upcast, with `safe_cast` and without it.

```
// safe_upcast.cpp
// compile with: /clr
using namespace System;
interface class A {
    void Test();
};

ref struct B : public A {
    virtual void Test() {
        Console::WriteLine("in B::Test");
    }

    void Test2() {
        Console::WriteLine("in B::Test2");
    }
};

ref struct C : public B {
    virtual void Test() override {
        Console::WriteLine("in C::Test");
    }
};

int main() {
    C ^ c = gcnew C;

    // implicit upcast
    B ^ b = c;
    b->Test();
    b->Test2();

    // upcast with safe_cast
    b = nullptr;
    b = safe_cast<B^>(c);
    b->Test();
    b->Test2();
}
```

```
in C::Test
in B::Test2
in C::Test
in B::Test2
```

## Downcasting

A downcast is a cast from a base class to a class that's derived from the base class. A downcast is safe only if the object that's addressed at runtime is actually addressing a derived class object. Unlike `static_cast`, `safe_cast` performs a dynamic check and throws `InvalidCastException` if the conversion fails.

```
// safe_downcast.cpp
// compile with: /clr
using namespace System;

interface class A { void Test(); };

ref struct B : public A {
    virtual void Test() {
        Console::WriteLine("in B::Test()");
    }

    void Test2() {
        Console::WriteLine("in B::Test2()");
    }
};

ref struct C : public B {
    virtual void Test() override {
        Console::WriteLine("in C::Test()");
    }
};

interface class I {};

value struct V : public I {};

int main() {
    A^ a = gcnew C();
    a->Test();
    B^ b = safe_cast<B^>(a);
    b->Test();
    b->Test2();

    V v;
    I^ i = v; // i boxes V
    V^ refv = safe_cast<V^>(i);

    Object^ o = gcnew B;
    A^ a2= safe_cast<A^>(o);
}
```

```
in C::Test()
in C::Test()
in B::Test2()
```

## safe\_cast with user-defined conversions

The next sample shows how you can use `safe_cast` to invoke user-defined conversions.

```

// safe_cast_udc.cpp
// compile with: /clr
using namespace System;
value struct V;

ref struct R {
    int x;
    R() {
        x = 1;
    }

    R(int argx) {
        x = argx;
    }

    static operator R::V^(R^ r);
};

value struct V {
    int x;
    static operator R^(V& v) {
        Console::WriteLine("in operator R^(V& v)");
        R^ r = gcnew R();
        r->x = v.x;
        return r;
    }

    V(int argx) {
        x = argx;
    }
};

R::operator V^(R^ r) {
    Console::WriteLine("in operator V^(R^ r)");
    return gcnew V(r->x);
}

int main() {
    bool fReturnVal = false;
    V v(2);
    R^ r = safe_cast<R^>(v); // should invoke UDC
    V^ v2 = safe_cast<V^>(r); // should invoke UDC
}

```

```

in operator R^(V& v
in operator V^(R^ r)

```

## safe\_cast and boxing operations

### Boxing

Boxing is defined as a compiler-injected, user-defined conversion. Therefore, you can use `safe_cast` to box a value on the CLR heap.

The following sample shows boxing with simple and user-defined value types. A `safe_cast` boxes a value type variable that's on the native stack so that it can be assigned to a variable on the garbage-collected heap.

```

// safe_cast_boxing.cpp
// compile with: /clr
using namespace System;

interface struct I {};

value struct V : public I {
    int m_x;

    V(int i) : m_x(i) {}
};

int main() {
    // box a value type
    V v(100);
    I^ i = safe_cast<I^>(v);

    int x = 100;
    V^ refv = safe_cast<V^>(v);
    int^ refi = safe_cast<int^>(x);
}

```

The next sample shows that boxing has priority over a user-defined conversion in a `safe_cast` operation.

```

// safe_cast_boxing_2.cpp
// compile with: /clr
static bool fRetVal = true;

interface struct I {};
value struct V : public I {
    int x;

    V(int argx) {
        x = argx;
    }

    static operator I^(V v) {
        fRetVal = false;
        I^ pi = v;
        return pi;
    }
};

ref struct R {
    R() {}
    R(V^ pv) {}
};

int main() {
    V v(10);
    I^ pv = safe_cast<I^>(v);    // boxing will occur, not UDC "operator I^"
}

```

## Unboxing

Unboxing is defined as a compiler-injected, user-defined conversion. Therefore, you can use `safe_cast` to unbox a value on the CLR heap.

Unboxing is a user-defined conversion, but unlike boxing, unboxing must be explicit—that is, it must be performed by a `static_cast`, C-style cast, or `safe_cast`; unboxing cannot be performed implicitly.

```
// safe_cast_unboxing.cpp
// compile with: /clr
int main() {
    System::Object ^ o = 42;
    int x = safe_cast<int>(o);
}
```

The following sample shows unboxing with value types and primitive types.

```
// safe_cast_unboxing_2.cpp
// compile with: /clr
using namespace System;

interface struct I {};

value struct VI : public I {};

void test1() {
    Object^ o = 5;
    int x = safe_cast<Int32>(o);
}

value struct V {
    int x;
    String^ s;
};

void test2() {
    V localv;
    Object^ o = localv;
    V unboxv = safe_cast<V>(o);
}

void test3() {
    V localv;
    V^ o2 = localv;
    V unboxv2 = safe_cast<V>(o2);
}

void test4() {
    I^ refi = VI();
    VI vi = safe_cast<VI>(refi);
}

int main() {
    test1();
    test2();
    test3();
    test4();
}
```

## safe\_cast and generic types

The next sample shows how you can use `safe_cast` to perform a downcast with a generic type.

```
// safe_cast_generic_types.cpp
// compile with: /clr
interface struct I {};

generic<class T> where T:I
ref struct Base {
    T t;
    void test1() {}
};

generic<class T> where T:I
ref struct Derived:public Base <T> {};

ref struct R:public I {};

typedef Base<R^> GBase_R;
typedef Derived<R^> GDerived_R;

int main() {
    GBase_R^ br = gcnew GDerived_R();
    GDerived_R^ dr = safe_cast<GDerived_R^>(br);
}
```

## See also

[safe\\_cast](#)

# Regular Expressions (C++/CLI)

9/20/2022 • 5 minutes to read • [Edit Online](#)

Demonstrates various string operations using regular expressions classes in the .NET Framework.

The following topics demonstrate the use of the .NET Framework [System.Text.RegularExpressions](#) namespace (and in one case the [System.String.Split](#) method) to search, parse, and modify strings.

## Parse Strings Using Regular Expressions

The following code example demonstrates simple string parsing using the [Regex](#) class in the [System.Text.RegularExpressions](#) namespace. A string containing multiple types of word delineators is constructed. The string is then parsed using the [Regex](#) class in conjunction with the [Match](#) class. Then, each word in the sentence is displayed separately.

### Example

```
// regex_parse.cpp
// compile with: /clr
#using <system.dll>

using namespace System;
using namespace System::Text::.RegularExpressions;

int main( )
{
    int words = 0;
    String^ pattern = "[a-zA-Z]*";
    Console::WriteLine( "pattern : '{0}'", pattern );
    Regex^ regex = gcnew Regex( pattern );

    String^ line = "one\ttwo three:four,five six  seven";
    Console::WriteLine( "text : '{0}'", line );
    for( Match^ match = regex->Match( line );
        match->Success; match = match->NextMatch( ) )
    {
        if( match->Value->Length > 0 )
        {
            words++;
            Console::WriteLine( "{0}", match->Value );
        }
    }
    Console::WriteLine( "Number of Words : {0}", words );

    return 0;
}
```

## Parse Strings Using the Split Method

The following code example demonstrates using the [System.String.Split](#) method to extract each word from a string. A string containing multiple types of word delineators is constructed and then parsed by calling [Split](#) with a list of the delineators. Then, each word in the sentence is displayed separately.

### Example

```

// regex_split.cpp
// compile with: /clr
using namespace System;

int main()
{
    String^ delimStr = " .:\t";
    Console::WriteLine( "delimiter : '{0}'", delimStr );
    array<Char>^ delimiter = delimStr->ToCharArray( );
    array<String>^ words;
    String^ line = "one\ttwo three:four,five six seven";

    Console::WriteLine( "text : '{0}'", line );
    words = line->Split( delimiter );
    Console::WriteLine( "Number of Words : {0}", words->Length );
    for (int word=0; word<words->Length; word++)
        Console::WriteLine( "{0}", words[word] );

    return 0;
}

```

## Use Regular Expressions for Simple Matching

The following code example uses regular expressions to look for exact substring matches. The search is performed by the static `IsMatch` method, which takes two strings as input. The first is the string to be searched, and the second is the pattern to be searched for.

### Example

```

// regex_simple.cpp
// compile with: /clr
#using <System.dll>

using namespace System;
using namespace System::Text::RegularExpressions;

int main()
{
    array<String>^ sentence =
    {
        "cow over the moon",
        "Betsy the Cow",
        "cowering in the corner",
        "no match here"
    };

    String^ matchStr = "cow";
    for (int i=0; i<sentence->Length; i++)
    {
        Console::Write( "{0,24}", sentence[i] );
        if ( Regex::.IsMatch( sentence[i], matchStr,
            RegexOptions::IgnoreCase ) )
            Console::WriteLine( " (match for '{0}' found)", matchStr );
        else
            Console::WriteLine("");
    }
    return 0;
}

```

## Use Regular Expressions to Extract Data Fields

The following code example demonstrates the use of regular expressions to extract data from a formatted string.

The following code example uses the [Regex](#) class to specify a pattern that corresponds to an e-mail address. This pattern includes field identifiers that can be used to retrieve the user and host name portions of each e-mail address. The [Match](#) class is used to perform the actual pattern matching. If the given e-mail address is valid, the user name and host names are extracted and displayed.

### Example

```
// Regex_extract.cpp
// compile with: /clr
#using <System.dll>

using namespace System;
using namespace System::Text::RegularExpressions;

int main()
{
    array<String^>^ address=
    {
        "jay@southridgevideo.com",
        "barry@adatum.com",
        "treyresearch.net",
        "karen@proseware.com"
    };

    Regex^ emailregex = gcnew Regex("(?<user>[^@]+)@(?:host)+");

    for (int i=0; i<address->Length; i++)
    {
        Match^ m = emailregex->Match( address[i] );
        Console::Write("\n{0,25}", address[i]);

        if ( m->Success )
        {
            Console::Write("    User='{0}'",
                m->Groups["user"]->Value);
            Console::Write("    Host='{0}'",
                m->Groups["host"]->Value);
        }
        else
            Console::Write("    (invalid email address)");
    }

    Console::WriteLine("");
    return 0;
}
```

## Use Regular Expressions to Rearrange Data

The following code example demonstrates how the .NET Framework regular expression support can be used to rearrange, or reformat data. The following code example uses the [Regex](#) and [Match](#) classes to extract first and last names from a string and then display these name elements in reverse order.

The [Regex](#) class is used to construct a regular expression that describes the current format of the data. The two names are assumed to be separated by a comma and can use any amount of white-space around the comma. The [Match](#) method is then used to analyze each string. If successful, first and last names are retrieved from the [Match](#) object and displayed.

### Example

```

// regex_reorder.cpp
// compile with: /clr
#using <System.dll>
using namespace System;
using namespace Text::RegularExpressions;

int main()
{
    array<String^>^ name =
    {
        "Abolrous, Sam",
        "Berg,Matt",
        "Berry , Jo",
        "www.contoso.com"
    };

    Regex^ reg = gcnew Regex("(?<last>\\w*)\\s*,\\s*(?<first>\\w*)");

    for ( int i=0; i < name->Length; i++ )
    {
        Console::Write( "{0,-20}", name[i] );
        Match^ m = reg->Match( name[i] );
        if ( m->Success )
        {
            String^ first = m->Groups["first"]->Value;
            String^ last = m->Groups["last"]->Value;
            Console::WriteLine("{0} {1}", first, last);
        }
        else
            Console::WriteLine("(invalid)");
    }
    return 0;
}

```

## Use Regular Expressions to Search and Replace

The following code example demonstrates how the regular expression class [Regex](#) can be used to perform search and replace. This is done with the [Replace](#) method. The version used takes two strings as input: the string to be modified, and the string to be inserted in place of the sections (if any) that match the pattern given to the [Regex](#) object.

This code replaces all the digits in a string with underscores (\_) and then replaces those with an empty string, effectively removing them. The same effect can be accomplished in a single step, but two steps are used here for demonstration purposes.

### Example

```
// regex_replace.cpp
// compile with: /clr
#using <System.dll>
using namespace System::Text::RegularExpressions;
using namespace System;

int main()
{
    String^ before = "The q43uick bro254wn f0ox ju4mped";
    Console::WriteLine("original : {0}", before);

    Regex^ digitRegex = gcnew Regex("(?<digit>[0-9])");
    String^ after = digitRegex->Replace(before, "_");
    Console::WriteLine("1st regex : {0}", after);

    Regex^ underbarRegex = gcnew Regex("_");
    String^ after2 = underbarRegex->Replace(after, "");
    Console::WriteLine("2nd regex : {0}", after2);

    return 0;
}
```

## Use Regular Expressions to Validate Data Formatting

The following code example demonstrates the use of regular expressions to verify the formatting of a string. In the following code example, the string should contain a valid phone number. The following code example uses the string "\d{3}-\d{3}-\d{4}" to indicate that each field represents a valid phone number. The "d" in the string indicates a digit, and the argument after each "d" indicates the number of digits that must be present. In this case, the number is required to be separated by dashes.

### Example

```
// regex_validate.cpp
// compile with: /clr
#using <System.dll>

using namespace System;
using namespace Text::RegularExpressions;

int main()
{
    array<String^>^ number =
    {
        "123-456-7890",
        "444-234-22450",
        "690-203-6578",
        "146-893-232",
        "146-839-2322",
        "4007-295-1111",
        "407-295-1111",
        "407-2-5555",
    };

    String^ regStr = "^\\d{3}-\\d{3}-\\d{4}$";

    for ( int i = 0; i < number->Length; i++ )
    {
        Console::Write( "{0,14}", number[i] );

        if ( Regex::IsMatch( number[i], regStr ) )
            Console::WriteLine(" - valid");
        else
            Console::WriteLine(" - invalid");
    }
    return 0;
}
```

## Related Sections

[.NET Framework Regular Expressions](#)

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

# File Handling and I/O (C++/CLI)

9/20/2022 • 5 minutes to read • [Edit Online](#)

Demonstrates various file operations using the .NET Framework.

The following topics demonstrate the use of classes defined in the [System.IO](#) namespace to perform various file operations.

## Enumerate Files in a Directory

The following code example demonstrates how to retrieve a list of the files in a directory. Additionally, the subdirectories are enumerated. The following code example uses the [GetFiles](#) and [GetDirectories](#) methods to display the contents of the C:\Windows directory.

### Example

```
// enum_files.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;

int main()
{
    String^ folder = "C:\\\\";
    array<String^>^ dir = Directory::GetDirectories( folder );
    Console::WriteLine("==== Directories inside '{0}' ====", folder);
    for (int i=0; i<dir->Length; i++)
        Console::WriteLine(dir[i]);

    array<String^>^ file = Directory::.GetFiles( folder );
    Console::WriteLine("==== Files inside '{0}' ====", folder);
    for (int i=0; i<file->Length; i++)
        Console::WriteLine(file[i]);

    return 0;
}
```

## Monitor File System Changes

The following code example uses [FileSystemWatcher](#) to register for events corresponding to files being created, changed, deleted, or renamed. Instead of periodically polling a directory for changes to files, you can use the [FileSystemWatcher](#) class to fire events when a change is detected.

### Example

```

// monitor_fs.cpp
// compile with: /clr
#using <system.dll>

using namespace System;
using namespace System::IO;

ref class FSEventHandler
{
public:
    void OnChanged (Object^ source, FileSystemEventArgs^ e)
    {
        Console::WriteLine("File: {0} {1}",
                           e->FullPath, e->ChangeType);
    }
    void OnRenamed(Object^ source, RenamedEventArgs^ e)
    {
        Console::WriteLine("File: {0} renamed to {1}",
                           e->OldFullPath, e->FullPath);
    }
};

int main()
{
    array<String^>^ args = Environment::GetCommandLineArgs();

    if(args->Length < 2)
    {
        Console::WriteLine("Usage: Watcher.exe <directory>");
        return -1;
    }

    FileSystemWatcher^ fsWatcher = gcnew FileSystemWatcher( );
    fsWatcher->Path = args[1];
    fsWatcher->NotifyFilter = static_cast<NotifyFilters>
        (NotifyFilters::FileName |
         NotifyFilters::Attributes |
         NotifyFilters::LastAccess |
         NotifyFilters::LastWrite |
         NotifyFilters::Security |
         NotifyFilters::Size );

    FSEventHandler^ handler = gcnew FSEventHandler();
    fsWatcher->Changed += gcnew FileSystemEventHandler(
        handler, &FSEventHandler::OnChanged);
    fsWatcher->Created += gcnew FileSystemEventHandler(
        handler, &FSEventHandler::OnChanged);
    fsWatcher->Deleted += gcnew FileSystemEventHandler(
        handler, &FSEventHandler::OnChanged);
    fsWatcher->Renamed += gcnew RenamedEventHandler(
        handler, &FSEventHandler::OnRenamed);

    fsWatcher->EnableRaisingEvents = true;

    Console::WriteLine("Press Enter to quit the sample.");
    Console::ReadLine( );
}

```

## Read a Binary File

The following code example shows how to read binary data from a file, by using two classes from the [System.IO](#) namespace: [FileStream](#) and [BinaryReader](#). [FileStream](#) represents the actual file. [BinaryReader](#) provides an interface to the stream that allows binary access.

The code example reads a file that's named data.bin and contains integers in binary format. For information

about this kind of file, see [How to: Write a Binary File \(C++/CLI\)](#).

### Example

```
// binary_read.cpp
// compile with: /clr
#using<system.dll>
using namespace System;
using namespace System::IO;

int main()
{
    String^ fileName = "data.bin";
    try
    {
        FileStream^ fs = gcnew FileStream(fileName, FileMode::Open);
        BinaryReader^ br = gcnew BinaryReader(fs);

        Console::WriteLine("contents of {0}:", fileName);
        while (br->BaseStream->Position < br->BaseStream->Length)
            Console::WriteLine(br->ReadInt32().ToString());

        fs->Close();
    }
    catch (Exception^ e)
    {
        if (dynamic_cast<FileNotFoundException^>(e))
            Console::WriteLine("File '{0}' not found", fileName);
        else
            Console::WriteLine("Exception: ({0})", e);
        return -1;
    }
    return 0;
}
```

## Read a Text File

The following code example demonstrates how to open and read a text file one line at a time, by using the [StreamReader](#) class that's defined in the [System.IO](#) namespace. An instance of this class is used to open a text file and then the [System.IO.StreamReader.ReadLine](#) method is used to retrieve each line.

This code example reads a file that's named `textfile.txt` and contains text. For information about this kind of file, see [How to: Write a Text File \(C++/CLI\)](#).

### Example

```

// text_read.cpp
// compile with: /clr
#using<system.dll>
using namespace System;
using namespace System::IO;

int main()
{
    String^ fileName = "textfile.txt";
    try
    {
        Console::WriteLine("trying to open file {0}...", fileName);
        StreamReader^ din = File::OpenText(fileName);

        String^ str;
        int count = 0;
        while ((str = din->ReadLine()) != nullptr)
        {
            count++;
            Console::WriteLine("line {0}: {1}", count, str );
        }
    }
    catch (Exception^ e)
    {
        if (dynamic_cast<FileNotFoundException^>(e))
            Console::WriteLine("file '{0}' not found", fileName);
        else
            Console::WriteLine("problem reading file '{0}'", fileName);
    }

    return 0;
}

```

## Retrieve File Information

The following code example demonstrates the [FileInfo](#) class. When you have the name of a file, you can use this class to retrieve information about the file such as the file size, directory, full name, and date and time of creation and of the last modification.

This code retrieves file information for Notepad.exe.

### Example

```
// file_info.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;

int main()
{
    array<String^>^ args = Environment::GetCommandLineArgs();
    if (args->Length < 2)
    {
        Console::WriteLine("\nUSAGE : file_info <filename>\n\n");
        return -1;
    }

    FileInfo^ fi = gcnew FileInfo( args[1] );

    Console::WriteLine("file size: {0}", fi->Length );

    Console::Write("File creation date:  ");
    Console::Write(fi->CreationTime.Month.ToString());
    Console::Write(".{0}", fi->CreationTime.Day.ToString());
    Console::WriteLine(".{0}", fi->CreationTime.Year.ToString());

    Console::Write("Last access date:  ");
    Console::Write(fi->>LastAccessTime.Month.ToString());
    Console::Write(".{0}", fi->>LastAccessTime.Day.ToString());
    Console::WriteLine(".{0}", fi->>LastAccessTime.Year.ToString());

    return 0;
}
```

## Write a Binary File

The following code example demonstrates writing binary data to a file. Two classes from the [System.IO](#) namespace are used: [FileStream](#) and [BinaryWriter](#). [FileStream](#) represents the actual file, while [BinaryWriter](#) provides an interface to the stream that allows binary access.

The following code example writes a file containing integers in binary format. This file can be read with the code in [How to: Read a Binary File \(C++/CLI\)](#).

### Example

```

// binary_write.cpp
// compile with: /clr
#using<system.dll>
using namespace System;
using namespace System::IO;

int main()
{
    array<Int32>^ data = {1, 2, 3, 10000};

    FileStream^ fs = gcnew FileStream("data.bin", FileMode::Create);
    BinaryWriter^ w = gcnew BinaryWriter(fs);

    try
    {
        Console::WriteLine("writing data to file:");
        for (int i=0; i<data->Length; i++)
        {
            Console::WriteLine(data[i]);
            w->Write(data[i]);
        }
    }
    catch (Exception^)
    {
        Console::WriteLine("data could not be written");
        fs->Close();
        return -1;
    }

    fs->Close();
    return 0;
}

```

## Write a Text File

The following code example demonstrates how to create a text file and write text to it using the [StreamWriter](#) class, which is defined in the [System.IO](#) namespace. The [StreamWriter](#) constructor takes the name of the file to be created. If the file exists, it is overwritten (unless you pass True as the second [StringWriter](#) constructor argument).

The file is then filed using the [Write](#) and [WriteLine](#) functions.

### Example

```
// text_write.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;

int main()
{
    String^ fileName = "textfile.txt";

    StreamWriter^ sw = gcnew StreamWriter(fileName);
    sw->WriteLine("A text file is born!");
    sw->Write("You can use WriteLine");
    sw->WriteLine("...or just Write");
    sw->WriteLine("and do {0} output too.", "formatted");
    sw->WriteLine("You can also send non-text objects:");
    sw->WriteLine(DateTime::Now);
    sw->Close();
    Console::WriteLine("a new file ('{0}') has been written", fileName);

    return 0;
}
```

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

[File and Stream I/O](#)

[System.IO namespace](#)

# Graphics Operations (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Demonstrates image manipulation using the Windows SDK.

The following topics demonstrate the use of the [System.Drawing.Image](#) class to perform image manipulation.

## Display Images with the .NET Framework

The following code example modifies the `OnPaint` event handler to retrieve a pointer to the [Graphics](#) object for the main form. The `OnPaint` function is intended for a Windows Forms application, most likely created with a Visual Studio application wizard.

The image is represented by the [Image](#) class. The image data is loaded from a jpg file using the [System.Drawing.Image.FromFile](#) method. Before the image is drawn to the form, the form is resized to accommodate the image. The drawing of the image is performed with the [System.Drawing.Graphics.DrawImage](#) method.

The [Graphics](#) and [Image](#) classes are both in the [System.Drawing](#) namespace.

### Example

```
#using <system.drawing.dll>

using namespace System;
using namespace System::Drawing;

protected:
virtual Void Form1::OnPaint(PaintEventArgs^ pe) override
{
    Graphics^ g = pe->Graphics;
    Image^ image = Image::FromFile("SampleImage.jpg");
    Form::ClientSize = image->Size;
    g->DrawImage( image, 0, 0, image->Size.Width, image->Size.Height );
}
```

## Draw Shapes with the .NET Framework

The following code example uses the [Graphics](#) class to modify the `OnPaint` event handler to retrieve a pointer to the [Graphics](#) object for the main form. This pointer is then used to set the background color of the form and draw a line and an arc using the [System.Drawing.Graphics.DrawLine](#) and [DrawArc](#) methods.

### Example

```

#using <system.drawing.dll>
using namespace System;
using namespace System::Drawing;
// ...
protected:
virtual Void Form1::OnPaint(PaintEventArgs^ pe ) override
{
    Graphics^ g = pe->Graphics;
    g->Clear(Color::AntiqueWhite);

    Rectangle rect = Form::ClientRectangle;
    Rectangle smallRect;
    smallRect.X = rect.X + rect.Width / 4;
    smallRect.Y = rect.Y + rect.Height / 4;
    smallRect.Width = rect.Width / 2;
    smallRect.Height = rect.Height / 2;

    Pen^ redPen = gcnew Pen(Color::Red);
    redPen->Width = 4;
    g->DrawLine(redPen, 0, 0, rect.Width, rect.Height);

    Pen^ bluePen = gcnew Pen(Color::Blue);
    bluePen->Width = 10;
    g->DrawArc( bluePen, smallRect, 90, 270 );
}

```

## Rotate Images with the .NET Framework

The following code example demonstrates the use of the [System.Drawing.Image](#) class to load an image from disk, rotate it 90 degrees, and save it as a new jpg file.

### Example

```

#using <system.drawing.dll>

using namespace System;
using namespace System::Drawing;

int main()
{
    Image^ image = Image::FromFile("SampleImage.jpg");
    image->RotateFlip( RotateFlipType::Rotate90FlipNone );
    image->Save("SampleImage_rotated.jpg");
    return 0;
}

```

## Convert Image File Formats with the .NET Framework

The following code example demonstrates the [System.Drawing.Image](#) class and the [System.Drawing.Imaging.ImageFormat](#) enumeration used to convert and save image files. The following code loads an image from a jpg file and then saves it in both .gif and .bmp file formats.

### Example

```
#using <system.drawing.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Drawing::Imaging;

int main()
{
    Image^ image = Image::FromFile("SampleImage.jpg");
    image->Save("SampleImage.png", ImageFormat::Png);
    image->Save("SampleImage.bmp", ImageFormat::Bmp);

    return 0;
}
```

## Related Sections

[Getting Started with Graphics Programming](#)

[About GDI+ Managed Code](#)

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

[System.Drawing](#)

# Windows Operations (C++/CLI)

9/20/2022 • 8 minutes to read • [Edit Online](#)

Demonstrates various Windows-specific tasks using the Windows SDK.

The following topics demonstrate various Windows operations performed with the Windows SDK using Visual C++.

## Determine if Shutdown Has Started

The following code example demonstrates how to determine whether the application or the .NET Framework is currently terminating. This is useful for accessing static elements in the .NET Framework because, during shutdown, these constructs are finalized by the system and cannot be reliably used. By checking the [HasShutdownStarted](#) property first, you can avoid potential failures by not accessing these elements.

### Example

```
// check_shutdown.cpp
// compile with: /clr
using namespace System;
int main()
{
    if (Environment::HasShutdownStarted)
        Console::WriteLine("Shutting down.");
    else
        Console::WriteLine("Not shutting down.");
    return 0;
}
```

## Determine the User Interactive State

The following code example demonstrates how to determine whether code is being run in a user-interactive context. If [UserInteractive](#) is false, then the code is running as a service process or from inside a Web application, in which case you should not attempt to interact with the user.

### Example

```
// user_interactive.cpp
// compile with: /clr
using namespace System;

int main()
{
    if ( Environment::UserInteractive )
        Console::WriteLine("User interactive");
    else
        Console::WriteLine("Noninteractive");
    return 0;
}
```

## Read Data from the Windows Registry

The following code example uses the [CurrentUser](#) key to read data from the Windows registry. First, the subkeys are enumerated using the [GetSubKeyNames](#) method and then the Identities subkey is opened using the

[OpenSubKey](#) method. Like the root keys, each subkey is represented by the [RegistryKey](#) class. Finally, the new [RegistryKey](#) object is used to enumerate the key/value pairs.

## Example

```
// registry_read.cpp
// compile with: /clr
using namespace System;
using namespace Microsoft::Win32;

int main( )
{
    array<String^>^ key = Registry::CurrentUser->GetSubKeyNames( );

    Console::WriteLine("Subkeys within CurrentUser root key:");
    for (int i=0; i<key->Length; i++)
    {
        Console::WriteLine("    {0}", key[i]);
    }

    Console::WriteLine("Opening subkey 'Identities'....");
    RegistryKey^ rk = nullptr;
    rk = Registry::CurrentUser->OpenSubKey("Identities");
    if (rk==nullptr)
    {
        Console::WriteLine("Registry key not found - aborting");
        return -1;
    }

    Console::WriteLine("Key/value pairs within 'Identities' key:");
    array<String^>^ name = rk->GetValueNames( );
    for (int i=0; i<name->Length; i++)
    {
        String^ value = rk->GetValue(name[i])->ToString();
        Console::WriteLine("    {0} = {1}", name[i], value);
    }

    return 0;
}
```

## Remarks

The [Registry](#) class is merely a container for static instances of [RegistryKey](#). Each instance represents a root registry node. The instances are [ClassesRoot](#), [CurrentConfig](#), [CurrentUser](#), [LocalMachine](#), and [Users](#).

In addition to being static, the objects within the [Registry](#) class are read-only. Furthermore, instances of the [RegistryKey](#) class that are created to access the contents of the registry objects are read-only as well. For an example of how to override this behavior, see [How to: Write Data to the Windows Registry \(C++/CLI\)](#).

There are two additional objects in the [Registry](#) class: [DynData](#) and [PerformanceData](#). Both are instances of the [RegistryKey](#) class. The [DynData](#) object contains dynamic registry information, which is only supported in Windows 98 and Windows Me. The [PerformanceData](#) object can be used to access performance counter information for applications that use the Windows Performance Monitoring System. The [PerformanceData](#) node represents information that is not actually stored in the registry and therefore cannot be viewed using Regedit.exe.

## Read Windows Performance Counters

Some applications and Windows subsystems expose performance data through the Windows performance system. These counters can be accessed using the [PerformanceCounterCategory](#) and [PerformanceCounter](#) classes, which reside in the [System.Diagnostics](#) namespace.

The following code example uses these classes to retrieve and display a counter that is updated by Windows to indicate the percentage of time that the processor is busy.

**NOTE**

This example requires administrative privileges to run on Windows Vista.

**Example**

```
// processor_timer.cpp
// compile with: /clr
#using <system.dll>

using namespace System;
using namespace System::Threading;
using namespace System::Diagnostics;
using namespace System::Timers;

ref struct TimerObject
{
public:
    static String^ m_instanceName;
    static PerformanceCounter^ m_theCounter;

public:
    static void OnTimer(Object^ source, ElapsedEventArgs^ e)
    {
        try
        {
            Console::WriteLine("CPU time used: {0,6} ",
                m_theCounter->NextValue( ).ToString("f"));
        }
        catch(Exception^ e)
        {
            if (dynamic_cast<InvalidOperationException^>(e))
            {
                Console::WriteLine("Instance '{0}' does not exist",
                    m_instanceName);
                return;
            }
            else
            {
                Console::WriteLine("Unknown exception... ('q' to quit)");
                return;
            }
        }
    }
};

int main()
{
    String^ objectName = "Processor";
    String^ counterName = "% Processor Time";
    String^ instanceName = "_Total";

    try
    {
        if ( !PerformanceCounterCategory::Exists(objectName) )
        {
            Console::WriteLine("Object {0} does not exist", objectName);
            return -1;
        }
    }
    catch (UnauthorizedAccessException ^ex)
    {
        Console::WriteLine("You are not authorized to access this information.");
    }
}
```

```

Console::WriteLine("You do not have permission to access this information.");
Console::Write("If you are using Windows Vista, run the application with ");
Console::WriteLine("administrative privileges.");
Console::WriteLine(ex->Message);
return -1;
}

if ( !PerformanceCounterCategory::CounterExists(
    counterName, objectName) )
{
    Console::WriteLine("Counter {0} does not exist", counterName);
    return -1;
}

TimerObject::m_instanceName = instanceName;
TimerObject::m_theCounter = gcnew PerformanceCounter(
    objectName, counterName, instanceName);

System::Timers::Timer^ aTimer = gcnew System::Timers::Timer();
aTimer->Elapsed += gcnew ElapsedEventHandler(&TimerObject::OnTimer);
aTimer->Interval = 1000;
aTimer->Enabled = true;
aTimer->AutoReset = true;

Console::WriteLine("reporting CPU usage for the next 10 seconds");
Thread::Sleep(10000);
return 0;
}

```

## Retrieve Text from the Clipboard

The following code example uses the [GetDataObject](#) member function to return a pointer to the [IDataObject](#) interface. This interface can then be queried for the format of the data and used to retrieve the actual data.

### Example

```

// read_clipboard.cpp
// compile with: /clr
#using <system.dll>
#using <system.Drawing.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Windows::Forms;

[STAThread] int main( )
{
    IDataObject^ data = Clipboard::GetDataObject( );

    if (data)
    {
        if (data->GetDataPresent(DataFormats::Text))
        {
            String^ text = static_cast<String^>
                (data->GetData(DataFormats::Text));
            Console::WriteLine(text);
        }
        else
            Console::WriteLine("Non-text data is in the Clipboard.");
    }
    else
    {
        Console::WriteLine("No data was found in the Clipboard.");
    }

    return 0;
}

```

## Retrieve the Current Username

The following code example demonstrates the retrieval of the current user name (the name of the user logged into Windows). The name is stored in the [UserName](#) string, which is defined in the [Environment](#) namespace.

### Example

```

// username.cpp
// compile with: /clr
using namespace System;

int main()
{
    Console::WriteLine("\nCurrent user: {0}", Environment::UserName);
    return 0;
}

```

## Retrieve the .NET Framework Version

The following code example demonstrates how to determine the version of the currently installed .NET Framework with the [Version](#) property, which is a pointer to a [Version](#) object that contains the version information.

### Example

```
// dotnet_ver.cpp
// compile with: /clr
using namespace System;
int main()
{
    Version^ version = Environment::Version;
    if (version)
    {
        int build = version->Build;
        int major = version->Major;
        int minor = version->Minor;
        int revision = Environment::Version->Revision;
        Console::Write(".NET Framework version: ");
        Console::WriteLine("{0}.{1}.{2}.{3}",
                           build, major, minor, revision);
    }
    return 0;
}
```

## Retrieve the Local Machine Name

The following code example demonstrates the retrieval of the local machine name (the name of the computer as it appears on a network). You can accomplish this by getting the [MachineName](#) string, which is defined in the [Environment](#) namespace.

### Example

```
// machine_name.cpp
// compile with: /clr
using namespace System;

int main()
{
    Console::WriteLine("\nMachineName: {0}", Environment::MachineName);
    return 0;
}
```

## Retrieve the Windows Version

The following code example demonstrates how to retrieve the platform and version information of the current operating system. This information is stored in the [System.Environment.OSVersion](#) property and consists of an enumeration that describes the version of Windows in broad terms and a [Version](#) object that contains the exact build of the operating system.

### Example

```

// os_ver.cpp
// compile with: /clr
using namespace System;

int main()
{
    OperatingSystem^ osv = Environment::OSVersion;
    PlatformID id = osv->Platform;
    Console::Write("Operating system: ");

    if (id == PlatformID::Win32NT)
        Console::WriteLine("Win32NT");
    else if (id == PlatformID::Win32S)
        Console::WriteLine("Win32S");
    else if (id == PlatformID::Win32Windows)
        Console::WriteLine("Win32Windows");
    else
        Console::WriteLine("WinCE");

    Version^ version = osv->Version;
    if (version)
    {
        int build = version->Build;
        int major = version->Major;
        int minor = version->Minor;
        int revision = Environment::Version->Revision;
        Console::Write("OS Version: ");
        Console::WriteLine("{0}.{1}.{2}.{3}",
                           build, major, minor, revision);
    }

    return 0;
}

```

## Retrieve Time Elapsed Since Startup

The following code example demonstrates how to determine the tick count, or number of milliseconds that have elapsed since Windows was started. This value is stored in the [System.Environment.TickCount](#) member and, because it is a 32-bit value, resets to zero approximately every 24.9 days.

### Example

```

// startup_time.cpp
// compile with: /clr
using namespace System;

int main( )
{
    Int32 tc = Environment::TickCount;
    Int32 seconds = tc / 1000;
    Int32 minutes = seconds / 60;
    float hours = static_cast<float>(minutes) / 60;
    float days = hours / 24;

    Console::WriteLine("Milliseconds since startup: {0}", tc);
    Console::WriteLine("Seconds since startup: {0}", seconds);
    Console::WriteLine("Minutes since startup: {0}", minutes);
    Console::WriteLine("Hours since startup: {0}", hours);
    Console::WriteLine("Days since startup: {0}", days);

    return 0;
}

```

## Store Text in the Clipboard

The following code example uses the [Clipboard](#) object defined in the [System.Windows.Forms](#) namespace to store a string. This object provides two member functions: [SetDataObject](#) and [GetDataObject](#). Data is stored in the Clipboard by sending any object derived from [Object](#) to [SetDataObject](#).

### Example

```
// store_clipboard.cpp
// compile with: /clr
#using <System.dll>
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

[STAThread] int main()
{
    String^ str = "This text is copied into the Clipboard.";

    // Use 'true' as the second argument if
    // the data is to remain in the clipboard
    // after the program terminates.
    Clipboard::SetDataObject(str, true);

    Console::WriteLine("Added text to the Clipboard.");

    return 0;
}
```

## Write Data to the Windows Registry

The following code example uses the [CurrentUser](#) key to create a writable instance of the [RegistryKey](#) class corresponding to the [Software](#) key. The [CreateSubKey](#) method is then used to create a new key and add to key/value pairs.

### Example

```

// registry_write.cpp
// compile with: /clr
using namespace System;
using namespace Microsoft::Win32;

int main()
{
    // The second OpenSubKey argument indicates that
    // the subkey should be writable.
    RegistryKey^ rk;
    rk = Registry::CurrentUser->OpenSubKey("Software", true);
    if (!rk)
    {
        Console::WriteLine("Failed to open CurrentUser/Software key");
        return -1;
    }

    RegistryKey^ nk = rk->CreateSubKey("NewRegKey");
    if (!nk)
    {
        Console::WriteLine("Failed to create 'NewRegKey'");
        return -1;
    }

    String^ newValue = "NewValue";
    try
    {
        nk->SetValue("NewKey", newValue);
        nk->SetValue("NewKey2", 44);
    }
    catch (Exception^)
    {
        Console::WriteLine("Failed to set new values in 'NewRegKey'");
        return -1;
    }

    Console::WriteLine("New key created.");
    Console::Write("Use REGEDIT.EXE to verify ");
    Console::WriteLine("'" + CURRENTUSER + "/Software/NewRegKey'\n");
    return 0;
}

```

## Remarks

You can use the .NET Framework to access the registry with the [Registry](#) and [RegistryKey](#) classes, which are both defined in the [Microsoft.Win32](#) namespace. The [Registry](#) class is a container for static instances of the [RegistryKey](#) class. Each instance represents a root registry node. The instances are [ClassesRoot](#), [CurrentConfig](#), [CurrentUser](#), [LocalMachine](#), and [Users](#).

## Related Sections

[Environment](#)

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

# Data Access Using ADO.NET (C++/CLI)

9/20/2022 • 16 minutes to read • [Edit Online](#)

ADO.NET is the .NET Framework API for data access and provides power and ease of use unmatched by previous data access solutions. This section describes some of the issues involving ADO.NET that are unique to Visual C++ users, such as marshaling native types.

ADO.NET runs under the Common Language Runtime (CLR). Therefore, any application that interacts with ADO.NET must also target the CLR. However, that does not mean that native applications cannot use ADO.NET. These examples will demonstrate how to interact with an ADO.NET database from native code.

## Marshal ANSI Strings for ADO.NET

Demonstrates how to add a native string (`char *`) to a database and how to marshal a `System.String` from a database to a native string.

### Example

In this example, the class `DatabaseClass` is created to interact with an ADO.NET `DataTable` object. Note that this class is a native C++ `class` (as compared to a `ref class` or `value class`). This is necessary because we want to use this class from native code, and you cannot use managed types in native code. This class will be compiled to target the CLR, as is indicated by the `#pragma managed` directive preceding the class declaration. For more information on this directive, see [managed, unmanaged](#).

Note the private member of the `DatabaseClass` class: `gcroot<DataTable ^> table`. Since native types cannot contain managed types, the `gcroot` keyword is necessary. For more information on `gcroot`, see [How to: Declare Handles in Native Types](#).

The rest of the code in this example is native C++ code, as is indicated by the `#pragma unmanaged` directive preceding `main`. In this example, we are creating a new instance of `DatabaseClass` and calling its methods to create a table and populate some rows in the table. Note that native C++ strings are being passed as values for the database column `StringCol`. Inside `DatabaseClass`, these strings are marshaled to managed strings using the marshaling functionality found in the `System.Runtime.InteropServices` namespace. Specifically, the method `PtrToStringAnsi` is used to marshal a `char *` to a `String`, and the method `StringToHGlobalAnsi` is used to marshal a `String` to a `char *`.

### NOTE

The memory allocated by `StringToHGlobalAnsi` must be deallocated by calling either `FreeHGlobal` or `GlobalFree`.

```
// adonet_marshal_string_native.cpp
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll
#include <comdef.h>
#include <gcroot.h>
#include <iostream>
using namespace std;

#using <System.Data.dll>
using namespace System;
using namespace System::Data;
using namespace System::Runtime::InteropServices;

#define MAXCOLS 100
```

```

#pragma managed
class DatabaseClass
{
public:
    DatabaseClass() : table(nullptr) { }

    void AddRow(char *stringValue)
    {
        // Add a row to the table.
        DataRow ^row = table->NewRow();
        row["StringCol"] = Marshal::PtrToStringAnsi(
            (IntPtr)stringValue);
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("StringCol",
            Type::GetType("System.String"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(char *dataColumn, char **values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringAnsi(
            (IntPtr)dataColumn);

        // Get all rows in the table.
        array<DataRow ^> ^rows = table->Select();
        int len = rows->Length;
        len = (len > valuesLength) ? valuesLength : len;
        for (int i = 0; i < len; i++)
        {
            // Marshal each column value from a managed string
            // to a char *.
            values[i] = (char *)Marshal::StringToHGlobalAnsi(
                (String ^)rows[i][columnStr]).ToPointer();
        }

        return len;
    }

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
    db->CreateAndPopulateTable();
    db->AddRow("This is string 1.");
    db->AddRow("This is string 2.");

    // Now retrieve the rows and display their contents.
    char *values[MAXCOLS];
    int len = db->GetValuesForColumn(
        "StringCol", values, MAXCOLS);
}

```

```

for (int i = 0; i < len; i++)
{
    cout << "StringCol: " << values[i] << endl;

    // Deallocate the memory allocated using
    // Marshal::StringToHGlobalAnsi.
    GlobalFree(values[i]);
}

delete db;

return 0;
}

```

```

StringCol: This is string 1.
StringCol: This is string 2.

```

## Compiling the Code

- To compile the code from the command line, save the code example in a file named adonet\_marshal\_string\_native.cpp and enter the following statement:

```
cl /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshal_string_native.cpp
```

## Marshal BSTR Strings for ADO.NET

Demonstrates how to add a COM string (`BSTR`) to a database and how to marshal a `System.String` from a database to a `BSTR`.

### Example

In this example, the class DatabaseClass is created to interact with an ADO.NET `DataTable` object. Note that this class is a native C++ `class` (as compared to a `ref class` or `value class`). This is necessary because we want to use this class from native code, and you cannot use managed types in native code. This class will be compiled to target the CLR, as is indicated by the `#pragma managed` directive preceding the class declaration. For more information on this directive, see [managed, unmanaged](#).

Note the private member of the DatabaseClass class: `gcroot<DataTable ^> table`. Since native types cannot contain managed types, the `gcroot` keyword is necessary. For more information on `gcroot`, see [How to: Declare Handles in Native Types](#).

The rest of the code in this example is native C++ code, as is indicated by the `#pragma unmanaged` directive preceding `main`. In this example, we are creating a new instance of DatabaseClass and calling its methods to create a table and populate some rows in the table. Note that COM strings are being passed as values for the database column StringCol. Inside DatabaseClass, these strings are marshaled to managed strings using the marshaling functionality found in the `System.Runtime.InteropServices` namespace. Specifically, the method `PtrToStringBSTR` is used to marshal a `BSTR` to a `String`, and the method `StringToBSTR` is used to marshal a `String` to a `BSTR`.

### NOTE

The memory allocated by `StringToBSTR` must be deallocated by calling either `FreeBSTR` or `SysFreeString`.

```

// adonet_marshal_string_bstr.cpp
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll
#include <comdef.h>

```

```

#include <gcroot.h>
#include <iostream>
using namespace std;

#using <System.Data.dll>
using namespace System;
using namespace System::Data;
using namespace System::Runtime::InteropServices;

#define MAXCOLUMNS 100

#pragma managed
class DatabaseClass
{
public:
    DatabaseClass() : table(nullptr) { }

    void AddRow(BSTR stringColValue)
    {
        // Add a row to the table.
        DataRow ^row = table->NewRow();
        row["StringCol"] = Marshal::PtrToStringBSTR(
            (IntPtr)stringColValue);
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("StringCol",
            Type::GetType("System.String"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(BSTR dataColumn, BSTR *values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringBSTR(
            (IntPtr)dataColumn);

        // Get all rows in the table.
        array<DataRow ^> ^rows = table->Select();
        int len = rows->Length;
        len = (len > valuesLength) ? valuesLength : len;
        for (int i = 0; i < len; i++)
        {
            // Marshal each column value from a managed string
            // to a BSTR.
            values[i] = (BSTR)Marshal::StringToBSTR(
                (String ^)rows[i][columnStr]).ToPointer();
        }

        return len;
    }

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{

```

```

// Create a table and add a few rows to it.
DatabaseClass *db = new DatabaseClass();
db->CreateAndPopulateTable();

BSTR str1 = SysAllocString(L"This is string 1.");
db->AddRow(str1);

BSTR str2 = SysAllocString(L"This is string 2.");
db->AddRow(str2);

// Now retrieve the rows and display their contents.
BSTR values[MAXCOLS];
BSTR str3 = SysAllocString(L"StringCol");
int len = db->GetValuesForColumn(
    str3, values, MAXCOLS);
for (int i = 0; i < len; i++)
{
    wcout << "StringCol: " << values[i] << endl;

    // Deallocate the memory allocated using
    // Marshal::StringToBSTR.
    SysFreeString(values[i]);
}

SysFreeString(str1);
SysFreeString(str2);
SysFreeString(str3);
delete db;

return 0;
}

```

```

StringCol: This is string 1.
StringCol: This is string 2.

```

## Compiling the Code

- To compile the code from the command line, save the code example in a file named `adonet_marshall_string_native.cpp` and enter the following statement:

```

cl /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshall_string_native.cpp

```

## Marshal Unicode Strings for ADO.NET

Demonstrates how to add a native Unicode string (`wchar_t *`) to a database and how to marshal a `System.String` from a database to a native Unicode string.

### Example

In this example, the class `DatabaseClass` is created to interact with an ADO.NET `DataTable` object. Note that this class is a native C++ `class` (as compared to a `ref class` or `value class`). This is necessary because we want to use this class from native code, and you cannot use managed types in native code. This class will be compiled to target the CLR, as is indicated by the `#pragma managed` directive preceding the class declaration. For more information on this directive, see [managed, unmanaged](#).

Note the private member of the `DatabaseClass` class: `gcroot<DataTable ^> table`. Since native types cannot contain managed types, the `gcroot` keyword is necessary. For more information on `gcroot`, see [How to: Declare Handles in Native Types](#).

The rest of the code in this example is native C++ code, as is indicated by the `#pragma unmanaged` directive

preceding `main`. In this example, we are creating a new instance of `DatabaseClass` and calling its methods to create a table and populate some rows in the table. Note that Unicode C++ strings are being passed as values for the database column `StringCol`. Inside `DatabaseClass`, these strings are marshaled to managed strings using the marshaling functionality found in the `System.Runtime.InteropServices` namespace. Specifically, the method `PtrToStringUni` is used to marshal a `wchar_t *` to a `String`, and the method `StringToHGlobalUni` is used to marshal a `String` to a `wchar_t *`.

#### NOTE

The memory allocated by `StringToHGlobalUni` must be deallocated by calling either `FreeHGlobal` or `GlobalFree`.

```
// adonet_marshal_string_wide.cpp
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll
#include <comdef.h>
#include <gcroot.h>
#include <iostream>
using namespace std;

#using <System.Data.dll>
using namespace System;
using namespace System::Data;
using namespace System::Runtime::InteropServices;

#define MAXCOLS 100

#pragma managed
class DatabaseClass
{
public:
    DatabaseClass() : table(nullptr) { }

    void AddRow(wchar_t *stringColValue)
    {
        // Add a row to the table.
        DataRow ^row = table->NewRow();
        row["StringCol"] = Marshal::PtrToStringUni(
            (IntPtr)stringColValue);
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("StringCol",
            Type::GetType("System.String"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(wchar_t *dataColumn, wchar_t **values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringUni(
            (IntPtr)dataColumn);

        // Get all rows in the table.
        array<DataRow ^> ^rows = table->Select();
        int len = rows->Length;
        len = (len > valuesLength) ? valuesLength : len;
        for (int i = 0; i < len; i++)
    }
```

```

    {
        // Marshal each column value from a managed string
        // to a wchar_t *.
        values[i] = (wchar_t *)Marshal::StringToHGlobalUni(
            (String ^)rows[i][columnStr]).ToPointer();
    }

    return len;
}

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
    db->CreateAndPopulateTable();
    db->AddRow(L"This is string 1.");
    db->AddRow(L"This is string 2.");

    // Now retrieve the rows and display their contents.
    wchar_t *values[MAXCOLS];
    int len = db->GetValuesForColumn(
        L"StringCol", values, MAXCOLS);
    for (int i = 0; i < len; i++)
    {
        wcout << "StringCol: " << values[i] << endl;

        // Deallocate the memory allocated using
        // Marshal::StringToHGlobalUni.
        GlobalFree(values[i]);
    }

    delete db;

    return 0;
}

```

```

StringCol: This is string 1.
StringCol: This is string 2.

```

## Compiling the Code

- To compile the code from the command line, save the code example in a file named adonet\_marshall\_string\_wide.cpp and enter the following statement:

```
c1 /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshall_string_wide.cpp
```

## Marshal a VARIANT for ADO.NET

Demonstrates how to add a native **VARIANT** to a database and how to marshal a **System.Object** from a database to a native **VARIANT**.

### Example

In this example, the class DatabaseClass is created to interact with an ADO.NET **DataTable** object. Note that this class is a native C++ **class** (as compared to a **ref class** or **value class**). This is necessary because we want

to use this class from native code, and you cannot use managed types in native code. This class will be compiled to target the CLR, as is indicated by the `#pragma managed` directive preceding the class declaration. For more information on this directive, see [managed, unmanaged](#).

Note the private member of the `DatabaseClass` class: `gcroot<DataTable ^> table`. Since native types cannot contain managed types, the `gcroot` keyword is necessary. For more information on `gcroot`, see [How to: Declare Handles in Native Types](#).

The rest of the code in this example is native C++ code, as is indicated by the `#pragma unmanaged` directive preceding `main`. In this example, we are creating a new instance of `DatabaseClass` and calling its methods to create a table and populate some rows in the table. Note that native `VARIANT` types are being passed as values for the database column `ObjectCol`. Inside `DatabaseClass`, these `VARIANT` types are marshaled to managed objects using the marshaling functionality found in the `System.Runtime.InteropServices` namespace. Specifically, the method `GetObjectForNativeVariant` is used to marshal a `VARIANT` to an `Object`, and the method `GetNativeVariantForObject` is used to marshal an `Object` to a `VARIANT`.

```
// adonet_marshal_variant.cpp
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll
#include <comdef.h>
#include <gcroot.h>
#include <iostream>
using namespace std;

#using <System.Data.dll>
using namespace System;
using namespace System::Data;
using namespace System::Runtime::InteropServices;

#define MAXCOLS 100

#pragma managed
class DatabaseClass
{
public:
    DatabaseClass() : table(nullptr) { }

    void AddRow(VARIANT *objectColValue)
    {
        // Add a row to the table.
        DataRow ^row = table->NewRow();
        row["ObjectCol"] = Marshal::GetObjectForNativeVariant(
            IntPtr(objectColValue));
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("ObjectCol",
            Type::GetType("System.Object"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(wchar_t *dataColumn, VARIANT *values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringUni(
            (IntPtr)dataColumn);
```

```

// Get all rows in the table.
array<DataRow ^> ^rows = table->Select();
int len = rows->Length;
len = (len > valuesLength) ? valuesLength : len;
for (int i = 0; i < len; i++)
{
    // Marshal each column value from a managed object
    // to a VARIANT.
    Marshal::GetNativeVariantForObject(
        rows[i][columnStr], IntPtr(&values[i]));
}

return len;
}

private:
// Using gcroot, you can use a managed type in
// a native class.
gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
    db->CreateAndPopulateTable();

    BSTR bstr1 = SysAllocString(L"This is a BSTR in a VARIANT.");
    VARIANT v1;
    v1.vt = VT_BSTR;
    v1.bstrVal = bstr1;
    db->AddRow(&v1);

    int i = 42;
    VARIANT v2;
    v2.vt = VT_I4;
    v2.lVal = i;
    db->AddRow(&v2);

    // Now retrieve the rows and display their contents.
    VARIANT values[MAXCOLS];
    int len = db->GetValuesForColumn(
        L"ObjectCol", values, MAXCOLS);
    for (int i = 0; i < len; i++)
    {
        switch (values[i].vt)
        {
            case VT_BSTR:
                wcout << L"ObjectCol: " << values[i].bstrVal << endl;
                break;
            case VT_I4:
                cout << "ObjectCol: " << values[i].lVal << endl;
                break;
            default:
                break;
        }
    }

    SysFreeString(bstr1);
    delete db;

    return 0;
}

```

```
ObjectCol: This is a BSTR in a VARIANT.  
ObjectCol: 42
```

## Compiling the Code

- To compile the code from the command line, save the code example in a file named `adonet_marshal_variant.cpp` and enter the following statement:

```
cl /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshal_variant.cpp
```

## Marshal a SAFEARRAY for ADO.NET

Demonstrates how to add a native `SAFEARRAY` to a database and how to marshal a managed array from a database to a native `SAFEARRAY`.

### Example

In this example, the class `DatabaseClass` is created to interact with an ADO.NET `DataTable` object. Note that this class is a native C++ `class` (as compared to a `ref class` or `value class`). This is necessary because we want to use this class from native code, and you cannot use managed types in native code. This class will be compiled to target the CLR, as is indicated by the `#pragma managed` directive preceding the class declaration. For more information on this directive, see [managed, unmanaged](#).

Note the private member of the `DatabaseClass` class: `gcroot<DataTable ^> table`. Since native types cannot contain managed types, the `gcroot` keyword is necessary. For more information on `gcroot`, see [How to: Declare Handles in Native Types](#).

The rest of the code in this example is native C++ code, as is indicated by the `#pragma unmanaged` directive preceding `main`. In this example, we are creating a new instance of `DatabaseClass` and calling its methods to create a table and populate some rows in the table. Note that native `SAFEARRAY` types are being passed as values for the database column `ArrayIntsCol`. Inside `DatabaseClass`, these `SAFEARRAY` types are marshaled to managed objects using the marshaling functionality found in the `System.Runtime.InteropServices` namespace. Specifically, the method `Copy` is used to marshal a `SAFEARRAY` to a managed array of integers, and the method `Copy` is used to marshal a managed array of integers to a `SAFEARRAY`.

```
// adonet_marshal_safearray.cpp  
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll  
#include <comdef.h>  
#include <gcroot.h>  
#include <iostream>  
using namespace std;  
  
#using <System.Data.dll>  
using namespace System;  
using namespace System::Data;  
using namespace System::Runtime::InteropServices;  
  
#define MAXCOLS 100  
  
#pragma managed  
class DatabaseClass  
{  
public:  
    DatabaseClass() : table(nullptr) { }  
  
    void AddRow(SAFEARRAY *arrayIntsColValue)  
    {  
        // Add a row to the table.  
        DataRow ^row = table->NewRow();
```

```

        int len = arrayIntsColValue->rgsabound[0].cElements;
        array<int> ^arr = gcnew array<int>(len);

        int *pData;
        SafeArrayAccessData(arrayIntsColValue, (void **)&pData);
        Marshal::Copy(IntPtr(pData), arr, 0, len);
        SafeArrayUnaccessData(arrayIntsColValue);

        row["ArrayIntsCol"] = arr;
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("ArrayIntsCol",
            Type::GetType("System.Int32[]"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(wchar_t *dataColumn, SAFEARRAY **values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringUni(
            (IntPtr)dataColumn);

        // Get all rows in the table.
        array<DataRow ^> ^rows = table->Select();
        int len = rows->Length;
        len = (len > valuesLength) ? valuesLength : len;
        for (int i = 0; i < len; i++)
        {
            // Marshal each column value from a managed array
            // of Int32s to a SAFEARRAY of type VT_I4.
            values[i] = SafeArrayCreateVector(VT_I4, 0, 10);
            int *pData;
            SafeArrayAccessData(values[i], (void **)&pData);
            Marshal::Copy((array<int> ^)rows[i][columnStr], 0,
                IntPtr(pData), 10);
            SafeArrayUnaccessData(values[i]);
        }

        return len;
    }

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
    db->CreateAndPopulateTable();

    // Create a standard array.
    int originalArray[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    // Create a SAFEARRAY.
    SAFEARRAY *psa;
    psa = SafeArrayCreateVector(VT_I4, 0, 10);
}

```

```

psa = SafeArrayCreateVector(v1_14, 0, 10);

// Copy the data from the original array to the SAFEARRAY.
int *pData;
HRESULT hr = SafeArrayAccessData(psa, (void **)&pData);
memcpy(pData, &originalArray, 40);
SafeArrayUnaccessData(psa);
db->AddRow(psa);

// Now retrieve the rows and display their contents.
SAFEARRAY *values[MAXCOLS];
int len = db->GetValuesForColumn(
    L"ArrayIntsCol", values, MAXCOLS);
for (int i = 0; i < len; i++)
{
    int *pData;
    SafeArrayAccessData(values[i], (void **)&pData);
    for (int j = 0; j < 10; j++)
    {
        cout << pData[j] << " ";
    }
    cout << endl;
    SafeArrayUnaccessData(values[i]);

    // Deallocate the memory allocated using
    // SafeArrayCreateVector.
    SafeArrayDestroy(values[i]);
}

SafeArrayDestroy(psa);
delete db;

return 0;
}

```

```
0 1 2 3 4 5 6 7 8 9
```

## Compiling the Code

- To compile the code from the command line, save the code example in a file named adonet\_marshall\_safearray.cpp and enter the following statement:

```
cl /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshall_safearray.cpp
```

## .NET Framework Security

For information on security issues involving ADO.NET, see [Securing ADO.NET Applications](#).

## Related Sections

SECTION	DESCRIPTION
<a href="#">ADO.NET</a>	Provides an overview of ADO.NET, a set of classes that expose data access services to the .NET programmer.

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

[Native and .NET Interoperability](#)

System.Runtime.InteropServices

Interoperability

# Native and .NET Interoperability

9/20/2022 • 2 minutes to read • [Edit Online](#)

Visual C++ supports interoperability features that allow managed and unmanaged constructs to co-exist and interoperate within the same assembly, and even in the same file. A small subset of this functionality, such as P/Invoke, is supported by other .NET languages as well, but most of the interoperability support provided by Visual C++ is not available in other languages.

## In This Section

### [Mixed \(Native and Managed\) Assemblies](#)

Describes assemblies generated with the [/clr \(Common Language Runtime Compilation\)](#) compiler option that contain both managed and unmanaged functionality.

### [Using a Windows Form User Control in MFC](#)

Discusses how to use the MFC Windows Forms support classes to host Windows Forms controls within your MFC applications.

### [Calling Native Functions from Managed Code](#)

Describes how non-CLR DLLs can be used from .NET applications.

# Interoperability with Other .NET Languages (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

The topics in this section show how to create assemblies in Visual C++ that consume from or provide functionality to assemblies written in C# or Visual Basic.

## Consume a C# Indexer

Visual C++ does not contain indexers; it has indexed properties. To consume a C# indexer, access the indexer as if it were an indexed property.

For more information about indexers, see:

- [Indexers](#)

### Example

The following C# program defines an indexer.

```
// consume_cs_indexers.cs
// compile with: /target:library
using System;
public class IndexerClass {
    private int [] myArray = new int[100];
    public int this [int index] {    // Indexer declaration
        get {
            // Check the index limits.
            if (index < 0 || index >= 100)
                return 0;
            else
                return myArray[index];
        }
        set {
            if (!(index < 0 || index >= 100))
                myArray[index] = value;
        }
    }
}
/*
// code to consume the indexer
public class MainClass {
    public static void Main() {
        IndexerClass b = new IndexerClass();

        // Call indexer to initialize elements 3 and 5
        b[3] = 256;
        b[5] = 1024;
        for (int i = 0 ; i <= 10 ; i++)
            Console.WriteLine("Element #{0} = {1}", i, b[i]);
    }
}
```

This C++/CLI program consumes the indexer.

```

// consume_cs_indexers_2.cpp
// compile with: /clr
#using "consume_cs_indexers.dll"
using namespace System;

int main() {
    IndexerClass ^ ic = gcnew IndexerClass;
    ic->default[0] = 21;
    for (int i = 0 ; i <= 10 ; i++)
        Console::WriteLine("Element #{0} = {1}", i, ic->default[i]);
}

```

The example produces this output:

```

Element #0 = 21
Element #1 = 0
Element #2 = 0
Element #3 = 0
Element #4 = 0
Element #5 = 0
Element #6 = 0
Element #7 = 0
Element #8 = 0
Element #9 = 0
Element #10 = 0

```

## Implement is and as C# Keywords

This topic shows how to implement the functionality of the `is` and `as` C# keywords in Visual C++.

### Example

```

// CS_is_as.cpp
// compile with: /clr
using namespace System;

interface class I {
public:
    void F();
};

ref struct C : public I {
    virtual void F( void ) { }
};

template < class T, class U >
Boolean isinst(U u) {
    return dynamic_cast< T >(u) != nullptr;
}

int main() {
    C ^ c = gcnew C();
    I ^ i = safe_cast< I ^ >(c);    // is (maps to castclass in IL)
    I ^ ii = dynamic_cast< I ^ >(c);    // as (maps to isinst in IL)

    // simulate 'as':
    Object ^ o = "f";
    if ( isinst< String ^ >(o) )
        Console::WriteLine("o is a string");
}

```

```
o is a string
```

## Implement the lock C# Keyword

This topic shows how to implement the C# `lock` keyword in Visual C++.

You can also use the `lock` class in the C++ Support Library. See [Synchronization \(lock Class\)](#) for more information.

### Example

```
// CS_lock_in_CPP.cpp
// compile with: /clr
using namespace System::Threading;
ref class Lock {
    Object^ m_pObject;
public:
    Lock( Object ^ pObject ) : m_pObject( pObject ) {
        Monitor::Enter( m_pObject );
    }
    ~Lock() {
        Monitor::Exit( m_pObject );
    }
};

ref struct LockHelper {
    void DoSomething();
};

void LockHelper::DoSomething() {
    // Note: Reference type with stack allocation semantics to provide
    // deterministic finalization

    Lock lock( this );
    // LockHelper instance is locked
}

int main()
{
    LockHelper lockHelper;
    lockHelper.DoSomething();
    return 0;
}
```

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

# Mixed (native and managed) assemblies

9/20/2022 • 2 minutes to read • [Edit Online](#)

Mixed assemblies are capable of containing both unmanaged machine instructions and MSIL instructions. This allows them to call and be called by .NET components, while retaining compatibility with native C++ libraries. Using mixed assemblies, developers can author applications using a mixture of .NET and native C++ code.

For example, an existing library consisting entirely of native C++ code can be brought to the .NET platform by recompiling just one module with the `/clr` compiler switch. This module is then able to use .NET features, but remains compatible with the remainder of the application. It is even possible to decide between managed and native compilation on a function-by-function basis within the same file (see [managed, unmanaged](#)).

Visual C++ only supports the generation of mixed managed assemblies by using the `/clr` compiler option. The `/clr:pure` and `/clr:safe` compiler options are deprecated in Visual Studio 2015 and unsupported in Visual Studio 2017. If you require pure or verifiable managed assemblies, we recommend you create them by using C#.

Earlier versions of the Microsoft C++ compiler toolset supported the generation of three distinct types of managed assemblies: mixed, pure, and verifiable. The latter two are discussed in [Pure and Verifiable Code \(C++/CLI\)](#).

## In this section

### [How to: Migrate to /clr](#)

Describes the recommended steps for either introducing or upgrading .NET functionality in your application.

### [How to: Compile MFC and ATL Code By Using /clr](#)

Discusses how to compile existing MFC and ATL programs to target the Common Language Runtime.

### [Initialization of Mixed Assemblies](#)

Describes the "loader lock" problem and solutions.

### [Library Support for Mixed Assemblies](#)

Discusses how to use native libraries in `/clr` compilations.

### [Performance Considerations](#)

Describes the performance implications of mixed assemblies and data marshaling.

### [Application Domains and Visual C++](#)

Discusses Visual C++ support for application domains.

### [Double Thunking](#)

Discusses the performance implications of a native entry point for a managed function.

### [Avoiding Exceptions on CLR Shutdown When Consuming COM Objects Built with /clr](#)

Discusses how to ensure proper shutdown of a managed application that consumes a COM object compiled with `/clr`.

### [How to: Create a Partially Trusted Application by Removing Dependency on the CRT Library DLL](#)

Discusses how to create a partially trusted Common Language Runtime application using Visual C++ by removing dependency on `msvcrt.dll`.

For more information about coding guidelines for mixed assemblies, see [An Overview of Managed/Unmanaged Code Interoperability](#).

## See also

- [Native and .NET Interoperability](#)

# How to: Migrate to /clr

9/20/2022 • 8 minutes to read • [Edit Online](#)

This topic discusses issues that arise when compiling native code with `/clr` (see [/clr \(Common Language Runtime Compilation\)](#) for more information). `/clr` allows native C++ code to invoke and be invoked from .NET assemblies in addition to other native C++ code. See [Mixed \(Native and Managed\) Assemblies](#) and [Native and .NET Interoperability](#) for more information on the advantages of compiling with `/clr`.

## Known Issues Compiling Library Projects with /clr

Visual Studio contains some known issues when compiling library projects with `/clr`:

- Your code may query types at runtime with `CRuntimeClass::FromName`. However, if a type is in an MSIL .dll (compiled with `/clr`), the call to `FromName` may fail if it occurs before the static constructors run in the managed .dll (you will not see this problem if the `FromName` call happens after code has executed in the managed .dll). To work around this problem, you can force the construction of the managed static constructor by defining a function in the managed .dll, exporting it, and invoking it from the native MFC application. For example:

```
// MFC extension DLL Header file:  
_declspec( dllexport ) void EnsureManagedInitialization () {  
    // managed code that won't be optimized away  
    System::GC::KeepAlive(System::Int32::.MaxValue);  
}
```

## Compile with Visual C++

Before using `/clr` on any module in your project, first compile and link your native project with Visual Studio 2010.

The following steps, followed in order, provide the easiest path to a `/clr` compilation. It is important to compile and run your project after each of these steps.

### Versions Prior to Visual Studio 2003

If you are upgrading to Visual Studio 2010 from a version prior to Visual Studio 2003, you may see compiler errors related to the enhanced C++ standard conformance in Visual Studio 2003.

### Upgrading from Visual Studio 2003

Projects previous built with Visual Studio 2003 should also first be compiled without `/clr` as Visual Studio now has increased ANSI/ISO conformance and some breaking changes. The change that is likely to require the most attention is [Security Features in the CRT](#). Code that uses the CRT is very likely to produce deprecation warnings. These warnings can be suppressed, but migrating to the new [Security-Enhanced Versions of CRT Functions](#) is preferred, as they provide better security and may reveal security issues in your code.

### Upgrading from Managed Extensions for C++

Starting in Visual Studio 2005, code written with Managed Extensions for C++ won't compile under `/clr`.

## Convert C Code to C++

Although Visual Studio will compile C files, it is necessary to convert them to C++ for a `/clr` compilation. The actual filename doesn't have to be changed; you can use `/Tp` (see [/Tc, /Tp, /TC, /TP \(Specify Source File Type.\)](#).)

Note that although C++ source code files are required for `/clr`, it is not necessary to re-factor your code to use object-oriented paradigms.

C code is very likely to require changes when compiled as a C++ file. The C++ type-safety rules are strict, so type conversions must be made explicit with casts. For example, malloc returns a void pointer, but can be assigned to a pointer to any type in C with a cast:

```
int* a = malloc(sizeof(int)); // C code
int* b = (int*)malloc(sizeof(int)); // C++ equivalent
```

Function pointers are also strictly type-safe in C++, so the following C code requires modification. In C++ it's best to create a `typedef` that defines the function pointer type, and then use that type to cast function pointers:

```
NewFunc1 = GetProcAddress( hLib, "Func1" ); // C code
typedef int(*MYPROC)(int); // C++ equivalent
NewFunc2 = (MYPROC)GetProcAddress( hLib, "Func2" );
```

C++ also requires that functions either be prototyped or fully defined before they can be referenced or invoked.

Identifiers used in C code that happen to be keywords in C++ (such as `virtual`, `new`, `delete`, `bool`, `true`, `false`, etc.) must be renamed. This can generally be done with simple search-and-replace operations.

```
COMObj1->lpVtbl->Method(COMObj, args); // C code
COMObj2->Method(args); // C++ equivalent
```

## Reconfigure Project Settings

After your project compiles and runs in Visual Studio 2010 you should create new project configurations for `/clr` rather than modifying the default configurations. `/clr` is incompatible with some compiler options and creating separate configurations lets you build your project as native or managed. When `/clr` is selected in the property pages dialog box, project settings not compatible with `/clr` are disabled (and disabled options are not automatically restored if `/clr` is subsequently unselected).

### Create New Project Configurations

You can use **Copy Settings From** option in the **New Project Configuration Dialog Box (Build > Configuration Manager > Active Solution Configuration > New)** to create a project configuration based on your existing project settings. Do this once for the Debug configuration and once for Release configuration. Subsequent changes can then be applied to the `/clr`-specific configurations only, leaving the original project configurations intact.

Projects that use custom build rules may require extra attention.

This step has different implications for projects that use makefiles. In this case a separate build-target can be configured, or version specific to `/clr` compilation can be created from a copy of the original.

### Change Project Settings

`/clr` can be selected in the development environment by following the instructions in [/clr \(Common Language Runtime Compilation\)](#). As mentioned previously, this step will automatically disable conflicting project settings.

#### **NOTE**

When upgrading a managed library or web service project from Visual Studio 2003, the `/ZI` compiler option will be added to the **Command Line** property page. This will cause LNK2001. Remove `/ZI` from the **Command Line** property page to resolve. See [/ZI \(Omit Default Library Name\)](#) and [Set compiler and build properties](#) for more information. Or, add `msvcrt.lib` and `msvcmt.lib` to the linker's **Additional Dependencies** property.

For projects built with makefiles, incompatible compiler options must be disabled manually once `/clr` is added. See [/clr Restrictions](#) for information on compiler options that are not compatible with `/clr`.

#### **Precompiled Headers**

Precompiled headers are supported under `/clr`. However, if you only compile some of your CPP files with `/clr` (compiling the rest as native) some changes will be required because precompiled headers generated with `/clr` are not compatible with those generated without `/clr`. This incompatibility is due to the fact that `/clr` generates and requires metadata. Modules compiled `/clr` can therefore not use precompiled headers that don't include metadata, and non `/clr` modules can't use precompiled header files that do contain meta data.

The easiest way to compile a project where some modules are compiled `/clr` is to disable precompiled headers entirely. (In the project Property Pages dialog, open the C/C++ node, and select Precompiled Headers. Then change the Create/Use Precompiled Headers property to "Not Using Precompiled Headers".)

However, particularly for large projects, precompiled headers provide much better compilation speed, so disabling this feature is not desirable. In this case it's best to configure the `/clr` and non `/clr` files to use separate precompiled headers. This can be done in one step by multi-selecting the modules to be compiled `/clr` using **Solution Explorer**, right-clicking on the group, and selecting Properties. Then change both the Create/Use PCH Through File and Precompiled Header File properties to use a different header file name and PCH file respectively.

## **Fixing Errors**

Compiling with `/clr` may result in compiler, linker or runtime errors. This section discusses the most common problems.

#### **Metadata Merge**

Differing versions of data types can cause the linker to fail because the metadata generated for the two types doesn't match. (This is usually caused when members of a type are conditionally defined, but the conditions are not the same for all CPP files that use the type.) In this case the linker fails, reporting only the symbol name and the name of the second OBJ file where the type was defined. It is often useful to rotate the order that OBJ files are sent to the linker to discover the location of the other version of the data type.

#### **Loader Lock Deadlock**

The "loader lock deadlock" can occur, but is deterministic and is detected and reported at runtime. See [Initialization of Mixed Assemblies](#) for detailed background, guidance, and solutions.

#### **Data Exports**

Exporting DLL data is error-prone, and not recommended. This is because the data section of a DLL is not guaranteed to be initialized until some managed portion of the DLL has been executed. Reference metadata with [#using Directive](#).

#### **Type Visibility**

Native types are private by default. This can result in a native type not being visible outside the DLL. Resolve this error by adding `public` to these types.

#### **Floating Point and Alignment Issues**

`__controlfp` is not supported on the common language runtime (see [\\_control87](#), [\\_controlfp](#), [\\_\\_control87\\_2](#) for more information). The CLR will also not respect `align`.

## COM Initialization

The Common Language Runtime initializes COM automatically when a module is initialized (when COM is initialized automatically it's done so as MTA). As a result, explicitly initializing COM yields return codes indicating that COM is already initialized. Attempting to explicitly initialize COM with one threading model when the CLR has already initialized COM to another threading model can cause your application to fail.

The common language runtime starts COM as MTA by default; use [/CLRTHREADATTRIBUTE \(Set CLR Thread Attribute\)](#) to modify this.

## Performance Issues

You may see decreased performance when native C++ methods generated to MSIL are called indirectly (virtual function calls or using function pointers). To learn more about this, see [Double Thunking](#).

When moving from native to MSIL, you will notice an increase in the size of your working set. This is because the common language runtime provides many features to ensure that programs run correctly. If your `/clr` application is not running correctly, you may want to enable C4793 (off by default), see [Compiler Warning \(level 1 and 3\) C4793](#) for more information.

## Program Crashes on Shutdown

In some cases, the CLR can shutdown before your managed code is finished running. Using `std::set_terminate` and `SIGTERM` can cause this. See [signal Constants](#) and [set\\_terminate](#) for more information.

# Using New Visual C++ Features

After your application compiles, links, and runs, you can begin using .NET features in any module compiled with `/clr`. For more information, see [Component Extensions for Runtime Platforms](#).

For information on .NET programming in Visual C++ see:

- [.NET Programming with C++/CLI \(Visual C++\)](#)
- [Native and .NET Interoperability](#)
- [Component Extensions for Runtime Platforms](#)

## See also

[Mixed \(Native and Managed\) Assemblies](#)

# How to: Compile MFC and ATL Code By Using /clr

9/20/2022 • 3 minutes to read • [Edit Online](#)

This topic discusses how to compile existing MFC and ATL programs to target the Common Language Runtime.

## To compile an MFC executable or regular MFC DLL by using /clr

1. Right-click the project in **Solution Explorer** and then click **Properties**.
2. In the **Project Properties** dialog box, expand the node next to **Configuration Properties** and select **General**. In the right pane, under **Project Defaults**, set **Common Language Runtime support** to **Common Language Runtime Support (/clr)**.  
In the same pane, make sure that **Use of MFC** is set to **Use MFC in a Shared DLL**.
3. Under **Configuration Properties**, expand the node next to **C/C++** and select **General**. Make sure that **Debug Information Format** is set to **Program Database /Zi** (not **/ZI**).
4. Select the **Code Generation** node. Set **Enable Minimal Rebuild** to **No (/Gm-)**. Also set **Basic Runtime Checks** to **Default**.
5. Under **Configuration Properties**, select **C/C++** and then **Code Generation**. Make sure that **Runtime Library** is set to either **Multi-threaded Debug DLL (/MDd)** or **Multi-threaded DLL (/MD)**.
6. In Stdafx.h, add the following line.

```
#using <System.Windows.Forms.dll>
```

## To compile an MFC extension DLL by using /clr

1. Follow the steps in "To compile an MFC executable or regular MFC DLL by using /clr".
2. Under **Configuration Properties**, expand the node next to **C/C++** and select **Precompiled Headers**. Set **Create/Use Precompiled Header** to **Not using Precompiled Headers**.

As an alternative, in **Solution Explorer**, right-click Stdafx.cpp and then click **Properties**. Under **Configuration Properties**, expand the node next to **C/C++** and select **General**. Set **Compile with Common Language Runtime support** to **No Common Language Runtime support**.

3. For the file that contains DllMain and anything it calls, in **Solution Explorer**, right-click the file and then click **Properties**. Under **Configuration Properties**, expand the node next to **C/C++** and select **General**. In the right pane, under **Project Defaults**, set **Compile with Common Language Runtime support** to **No Common Language Runtime support**.

## To compile an ATL executable by using /clr

1. In **Solution Explorer**, right-click the project and then click **Properties**.
2. In the **Project Properties** dialog box, expand the node next to **Configuration Properties** and select **General**. In the right pane, under **Project Defaults**, set **Common Language Runtime support** to **Common Language Runtime Support (/clr)**.
3. Under **Configuration Properties**, expand the node next to **C/C++** and select **General**. Make sure that **Debug Information Format** is set to **Program Database /Zi** (not **/ZI**).
4. Select the **Code Generation** node. Set **Enable Minimal Rebuild** to **No (/Gm-)**. Also set **Basic**

## **Runtime Checks to Default.**

5. Under **Configuration Properties**, select **C/C++** and then **Code Generation**. Make sure that **Runtime Library** is set to either **Multi-threaded Debug DLL (/MDd)** or **Multi-threaded DLL (/MD)**.
6. For every MIDL-generated file (C files), right-click the file in **Solution Explorer** and then click **Properties**. Under **Configuration Properties**, expand the node next to **C/C++** and select **General**. Set **Compile with Common Language Runtime support** to **No Common Language Runtime support**.

## **To compile an ATL DLL by using /clr**

1. Follow the steps in the "To compile an ATL executable by using /clr" section.
2. Under **Configuration Properties**, expand the node next to **C/C++** and select **Precompiled Headers**. Set **Create/Use Precompiled Header** to **Not using Precompiled Headers**.

As an alternative, in **Solution Explorer**, right-click **Stdafx.cpp** and then click **Properties**. Under **Configuration Properties**, expand the node next to **C/C++** and select **General**. Set **Compile with Common Language Runtime support** to **No Common Language Runtime support**.

3. For the file that contains **DllMain** and anything it calls, in **Solution Explorer**, right-click the file and then click **Properties**. Under **Configuration Properties**, expand the node next to **C/C++** and select **General**. In the right pane, under **Project Defaults**, set **Compile with Common Language Runtime support** to **No Common Language Runtime support**.

## **See also**

[Mixed \(Native and Managed\) Assemblies](#)

# Initialization of Mixed Assemblies

9/20/2022 • 13 minutes to read • [Edit Online](#)

Windows developers must always be wary of loader lock when running code during `DllMain`. However, there are some additional issues to consider when dealing with C++/CLI mixed-mode assemblies.

Code within `DllMain` must not access the .NET Common Language Runtime (CLR). That means that `DllMain` should make no calls to managed functions, directly or indirectly; no managed code should be declared or implemented in `DllMain`; and no garbage collection or automatic library loading should take place within `DllMain`.

## Causes of Loader Lock

With the introduction of the .NET platform, there are two distinct mechanisms for loading an execution module (EXE or DLL): one for Windows, which is used for unmanaged modules, and one for the CLR, which loads .NET assemblies. The mixed DLL loading problem centers around the Microsoft Windows OS loader.

When an assembly containing only .NET constructs is loaded into a process, the CLR loader can do all of the necessary loading and initialization tasks itself. However, to load mixed assemblies that can contain native code and data, the Windows loader must be used as well.

The Windows loader guarantees that no code can access code or data in that DLL before it's been initialized. And it ensures that no code can redundantly load the DLL while it's partially initialized. To do it, the Windows loader uses a process-global critical section (often called the "loader lock") that prevents unsafe access during module initialization. As a result, the loading process is vulnerable to many classic deadlock scenarios. For mixed assemblies, the following two scenarios increase the risk of deadlock:

- First, if users attempt to execute functions compiled to Microsoft intermediate language (MSIL) when the loader lock is held (from `DllMain` or in static initializers, for example), it can cause deadlock. Consider the case in which the MSIL function references a type in an assembly that's not loaded yet. The CLR will attempt to automatically load that assembly, which may require the Windows loader to block on the loader lock. A deadlock occurs, since the loader lock is already held by code earlier in the call sequence. However, executing MSIL under loader lock doesn't guarantee that a deadlock will occur. That's what makes this scenario difficult to diagnose and fix. In some circumstances, such as when the DLL of the referenced type contains no native constructs and all of its dependencies contain no native constructs, the Windows loader isn't required to load the .NET assembly of the referenced type. Additionally, the required assembly or its mixed native/.NET dependencies may have already been loaded by other code. Consequently, the deadlocking can be difficult to predict, and can vary depending on the configuration of the target machine.
- Second, when loading DLLs in versions 1.0 and 1.1 of the .NET Framework, the CLR assumed that the loader lock wasn't held, and took several actions that are invalid under loader lock. Assuming that the loader lock isn't held is a valid assumption for purely .NET DLLs. But because mixed DLLs execute native initialization routines, they require the native Windows loader, and consequently the loader lock. So, even if the developer wasn't attempting to execute any MSIL functions during DLL initialization, there was still a small possibility of nondeterministic deadlock in .NET Framework versions 1.0 and 1.1.

All non-determinism has been removed from the mixed DLL loading process. It was accomplished with these changes:

- The CLR no longer makes false assumptions when loading mixed DLLs.

- Unmanaged and managed initialization is done in two separate and distinct stages. Unmanaged initialization takes place first (via `DllMain`), and managed initialization takes place afterwards, through a .NET-supported `.cctor` construct. The latter is completely transparent to the user unless `/Z1` or `/NODEFAULTLIB` are used. For more information, see [/NODEFAULTLIB \(Ignore Libraries\)](#) and [/Z1 \(Omit Default Library Name\)](#).

Loader lock can still occur, but now it occurs reproducibly, and is detected. If `DllMain` contains MSIL instructions, the compiler generates warning [Compiler Warning \(level 1\) C4747](#). Furthermore, either the CRT or the CLR will try to detect and report attempts to execute MSIL under loader lock. CRT detection results in runtime diagnostic C Run-Time Error R6033.

The rest of this article describes the remaining scenarios for which MSIL can execute under the loader lock. It shows how to resolve the issue under each of those scenarios, and debugging techniques.

## Scenarios and Workarounds

There are several different situations under which user code can execute MSIL under loader lock. The developer must ensure that the user code implementation doesn't attempt to execute MSIL instructions under each of these circumstances. The following subsections describe all possibilities with a discussion of how to resolve issues in the most common cases.

### `DllMain`

The `DllMain` function is a user-defined entry point for a DLL. Unless the user specifies otherwise, `DllMain` is invoked every time a process or thread attaches to or detaches from the containing DLL. Since this invocation can occur while the loader lock is held, no user-supplied `DllMain` function should be compiled to MSIL. Furthermore, no function in the call tree rooted at `DllMain` can be compiled to MSIL. To resolve issues here, the code block that defines `DllMain` should be modified with `#pragma unmanaged`. The same should be done for every function that `DllMain` calls.

In cases where these functions must call a function that requires an MSIL implementation for other calling contexts, you can use a duplication strategy where both a .NET and a native version of the same function are created.

As an alternative, if `DllMain` isn't required or if it doesn't need to be executed under loader lock, you can remove the user-provided `DllMain` implementation, which eliminates the problem.

If `DllMain` attempts to execute MSIL directly, [Compiler Warning \(level 1\) C4747](#) will result. However, the compiler can't detect cases where `DllMain` calls a function in another module that in turn attempts to execute MSIL.

For more information on this scenario, see [Impediments to Diagnosis](#).

### Initializing Static Objects

Initializing static objects can result in deadlock if a dynamic initializer is required. Simple cases (such as when you assign a value known at compile time to a static variable) don't require dynamic initialization, so there's no risk of deadlock. However, some static variables get initialized by function calls, constructor invocations, or expressions that can't be evaluated at compile time. These variables all require code to execute during module initialization.

The code below shows examples of static initializers that require dynamic initialization: a function call, object construction, and a pointer initialization. (These examples aren't static, but are assumed to have definitions in the global scope, which has the same effect.)

```
// dynamic initializer function generated
int a = init();
CObject o(arg1, arg2);
CObject* op = new CObject(arg1, arg2);
```

This risk of deadlock depends on whether the containing module is compiled with `/clr` and whether MSIL will be executed. Specifically, if the static variable is compiled without `/clr` (or is in a `#pragma unmanaged` block), and the dynamic initializer required to initialize it results in the execution of MSIL instructions, deadlock may occur. It's because, for modules compiled without `/clr`, the initialization of static variables is performed by DllMain. In contrast, static variables compiled with `/clr` are initialized by the `.cctor`, after the unmanaged initialization stage has completed and the loader lock has been released.

There are a number of solutions to deadlock caused by the dynamic initialization of static variables. They're arranged here roughly in order of time required to fix the problem:

- The source file containing the static variable can be compiled with `/clr`.
- All functions called by the static variable can be compiled to native code using the `#pragma unmanaged` directive.
- Manually clone the code that the static variable depends upon, providing both a .NET and a native version with different names. Developers can then call the native version from native static initializers and call the .NET version elsewhere.

### User-Supplied Functions Affecting Startup

There are several user-supplied functions on which libraries depend for initialization during startup. For example, when globally overloading operators in C++ such as the `new` and `delete` operators, the user-provided versions are used everywhere, including in C++ Standard Library initialization and destruction. As a result, C++ Standard Library and user-provided static initializers will invoke any user-provided versions of these operators.

If the user-provided versions are compiled to MSIL, then these initializers will be attempting to execute MSIL instructions while the loader lock is held. A user-supplied `malloc` has the same consequences. To resolve this problem, any of these overloads or user-supplied definitions must be implemented as native code using the `#pragma unmanaged` directive.

For more information on this scenario, see [Impediments to Diagnosis](#).

### Custom Locales

If the user provides a custom global locale, this locale gets used to initialize all future I/O streams, including streams that are statically initialized. If this global locale object is compiled to MSIL, then locale-object member functions compiled to MSIL may be invoked while the loader lock is held.

There are three options for solving this problem:

The source files containing all global I/O stream definitions can be compiled using the `/clr` option. It prevents their static initializers from being executed under loader lock.

The custom locale function definitions can be compiled to native code by using the `#pragma unmanaged` directive.

Refrain from setting the custom locale as the global locale until after the loader lock is released. Then explicitly configure I/O streams created during initialization with the custom locale.

## Impediments to Diagnosis

In some cases, it's difficult to detect the source of deadlocks. The following subsections discuss these scenarios

and ways to work around these issues.

## Implementation in Headers

In select cases, function implementations inside header files can complicate diagnosis. Inline functions and template code both require that functions be specified in a header file. The C++ language specifies the One Definition Rule, which forces all implementations of functions with the same name to be semantically equivalent. Consequently, the C++ linker need not make any special considerations when merging object files that have duplicate implementations of a given function.

In Visual Studio versions before Visual Studio 2005, the linker simply chooses the largest of these semantically equivalent definitions. It's done to accommodate forward declarations, and scenarios when different optimization options are used for different source files. It creates a problem for mixed native and .NET DLLs.

Because the same header may be included both by C++ files with `/clr` enabled and disabled, or a `#include` can be wrapped inside a `#pragma unmanaged` block, it's possible to have both MSIL and native versions of functions that provide implementations in headers. MSIL and native implementations have different semantics for initialization under the loader lock, which effectively violates the one definition rule. Consequently, when the linker chooses the largest implementation, it may choose the MSIL version of a function, even if it was explicitly compiled to native code elsewhere using the `#pragma unmanaged` directive. To ensure that an MSIL version of a template or inline function is never called under loader lock, every definition of every such function called under loader lock must be modified with the `#pragma unmanaged` directive. If the header file is from a third party, the easiest way to make this change is to push and pop the `#pragma unmanaged` directive around the `#include` directive for the offending header file. (See [managed, unmanaged](#) for an example.) However, this strategy doesn't work for headers that contain other code that must directly call .NET APIs.

As a convenience for users dealing with loader lock, the linker will choose the native implementation over the managed when presented with both. This default avoids the above issues. However, there are two exceptions to this rule in this release because of two unresolved issues with the compiler:

- The call to an inline function is through a global static function pointer. This scenario isn'table because virtual functions are called through global function pointers. For example,

```
#include "definesmyObject.h"
#include "definesclassC.h"

typedef void (*function_pointer_t)();

function_pointer_t myObject_p = &myObject;

#pragma unmanaged
void DuringLoaderlock(C & c)
{
    // Either of these calls could resolve to a managed implementation,
    // at link-time, even if a native implementation also exists.
    c.VirtualMember();
    myObject_p();
}
```

## Diagnosing in Debug Mode

All diagnoses of loader lock problems should be done with Debug builds. Release builds may not produce diagnostics. And, the optimizations made in Release mode may mask some of the MSIL under loader lock scenarios.

## How to debug loader lock issues

The diagnostic that the CLR generates when an MSIL function is invoked causes the CLR to suspend execution. That in turn causes the Visual C++ mixed-mode debugger to be suspended as well when running the debuggee

in-process. However, when attaching to the process, it'sn't possible to obtain a managed callstack for the debuggee using the mixed debugger.

To identify the specific MSIL function that was called under loader lock, developers should complete the following steps:

1. Ensure that symbols for mscoree.dll and mscorewks.dll are available.

You can make the symbols available in two ways. First, the PDBs for mscoree.dll and mscorewks.dll can be added to the symbol search path. To add them, open the symbol search path options dialog. (From the **Tools** menu, choose **Options**. In the left pane of the **Options** dialog box, open the **Debugging** node and choose **Symbols**.) Add the path to the mscoree.dll and mscorewks.dll PDB files to the search list. These PDBs are installed to the %VSINSTALLDIR%\SDK\v2.0\symbols. Choose **OK**.

Second, the PDBs for mscoree.dll and mscorewks.dll can be downloaded from the Microsoft Symbol Server. To configure Symbol Server, open the symbol search path options dialog. (From the **Tools** menu, choose **Options**. In the left pane of the **Options** dialog box, open the **Debugging** node and choose **Symbols**.) Add this search path to the search list: <https://msdl.microsoft.com/download/symbols>. Add a symbol cache directory to the symbol server cache text box. Choose **OK**.

2. Set debugger mode to native-only mode.

Open the **Properties** grid for the startup project in the solution. Select **Configuration Properties** > **Debugging**. Set the **Debugger Type** property to **Native-Only**.

3. Start the debugger (F5).

4. When the `/clr` diagnostic is generated, choose **Retry** and then choose **Break**.

5. Open the call stack window. (On the menu bar, choose **Debug** > **Windows** > **Call Stack**.) The offending `DllMain` or static initializer is identified with a green arrow. If the offending function isn't identified, the following steps must be taken to find it.

6. Open the **Immediate** window (On the menu bar, choose **Debug** > **Windows** > **Immediate**.)

7. Enter `.load sos.dll` into the **Immediate** window to load the SOS debugging service.

8. Enter `!dumpstack` into the **Immediate** window to obtain a complete listing of the internal `/clr` stack.

9. Look for the first instance (closest to the bottom of the stack) of either `_CorDlIMain` (if `DllMain` causes the issue) or `_VTableBootstrapThunkInitHelperStub` or `GetTargetForVTableEntry` (if a static initializer causes the issue). The stack entry just below this call is the invocation of the MSIL implemented function that attempted to execute under loader lock.

10. Go to the source file and line number identified in the previous step and correct the problem using the scenarios and solutions described in the Scenarios section.

## Example

### Description

The following sample shows how to avoid loader lock by moving code from `DllMain` into the constructor of a global object.

In this sample, there's a global managed object whose constructor contains the managed object that was originally in `DllMain`. The second part of this sample references the assembly, creating an instance of the managed object to invoke the module constructor that does the initialization.

### Code

```

// initializing_mixed_assemblies.cpp
// compile with: /clr /LD
#pragma once
#include <stdio.h>
#include <windows.h>
struct __declspec(dllexport) A {
    A() {
        System::Console::WriteLine("Module ctor initializing based on global instance of class.\n");
    }

    void Test() {
        printf_s("Test called so linker doesn't throw away unused object.\n");
    }
};

#pragma unmanaged
// Global instance of object
A obj;

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved) {
    // Remove all managed code from here and put it in constructor of A.
    return true;
}

```

This example demonstrates issues in initialization of mixed assemblies:

```

// initializing_mixed_assemblies_2.cpp
// compile with: /clr initializing_mixed_assemblies.lib
#include <windows.h>
using namespace System;
#include <stdio.h>
#using "initializing_mixed_assemblies.dll"
struct __declspec(dllimport) A {
    void Test();
};

int main() {
    A obj;
    obj.Test();
}

```

This code produces the following output:

```

Module ctor initializing based on global instance of class.

Test called so linker doesn't throw away unused object.

```

## See also

[Mixed \(Native and Managed\) Assemblies](#)

# Library Support for Mixed Assemblies

9/20/2022 • 2 minutes to read • [Edit Online](#)

Visual C++ supports the use of the C++ Standard Library, the C runtime library (CRT), ATL, and MFC for applications compiled with [/clr \(Common Language Runtime Compilation\)](#). This allows existing applications that use these libraries to use .NET Framework features as well.

## IMPORTANT

The `/clr:pure` and `/clr:safe` compiler options are deprecated in Visual Studio 2015 and unsupported in Visual Studio 2017.

This support includes the following DLL and import libraries:

- `Msvcmrt[d].lib` if you compile with `/clr`. Mixed assemblies link to this import library.

This support provides several related benefits:

- The CRT and C++ Standard Library are available to mixed code. The CRT and C++ Standard Library provided are not verifiable; ultimately, your calls are still routed to the same CRT and C++ Standard Library as you are using from native code.
- Correct unified exception handling in mixed images.
- Static initialization of C++ variables in mixed images.
- Support for per-`AppDomain` and per-process variables in managed code.
- Resolves the loader lock issues that applied to mixed DLLs compiled in Visual Studio 2003 and earlier.

In addition, this support presents the following limitations:

- Only the CRT DLL model is supported for code compiled with `/clr`. There are no static CRT libraries that support `/clr` builds.

## See also

- [Mixed \(Native and Managed\) Assemblies](#)

# Performance Considerations for Interop (C++)

9/20/2022 • 3 minutes to read • [Edit Online](#)

This topic provides guidelines for reducing the effect of managed/unmanaged interop transitions on run-time performance.

Visual C++ supports the same interoperability mechanisms as other .NET languages such as Visual Basic and C# (P/Invoke), but it also provides interop support that is specific to Visual C++ (C++ interop). For performance-critical applications, it is important to understand the performance implications of each interop technique.

Regardless of the interop technique used, special transition sequences, called thunks, are required each time a managed function calls an unmanaged function and vice versa. These thunks are inserted automatically by the Microsoft C++ compiler, but it is important to keep in mind that cumulatively, these transitions can be expensive in terms of performance.

## Reducing Transitions

One way to avoid or reduce the cost of interop thunks is to refactor the interfaces involved to minimize managed/unmanaged transitions. Dramatic performance improvements can be made by targeting chatty interfaces, which are those that involve frequent calls across the managed/unmanaged boundary. A managed function that calls an unmanaged function in a tight loop, for example, is a good candidate for refactoring. If the loop itself is moved to the unmanaged side, or if a managed alternative to the unmanaged call is created (perhaps by queuing data on the managed side and then marshaling it to the unmanaged API all at once after the loop), the number of transitions can be reduced significantly.

## P/Invoke vs. C++ Interop

For .NET languages, such as Visual Basic and C#, the prescribed method for interoperating with native components is P/Invoke. Because P/Invoke is supported by the .NET Framework, Visual C++ supports it as well, but Visual C++ also provides its own interoperability support, which is referred to as C++ Interop. C++ Interop is preferred over P/Invoke because P/Invoke is not type-safe. As a result, errors are primarily reported at run time, but C++ Interop also has performance advantages over P/Invoke.

Both techniques require several things to happen whenever a managed function calls an unmanaged function:

- The function call arguments are marshaled from CLR to native types.
- A managed-to-unmanaged thunk is executed.
- The unmanaged function is called (using the native versions of the arguments).
- An unmanaged-to-managed thunk is executed.
- The return type and any "out" or "in,out" arguments are marshaled from native to CLR types.

The managed/unmanaged thunks are necessary for interop to work at all, but the data marshaling that is required depends on the data types involved, the function signature, and how the data will be used.

The data marshaling performed by C++ Interop is the simplest possible form: the parameters are simply copied across the managed/unmanaged boundary in a bitwise fashion; no transformation is performed at all. For P/Invoke, this is only true if all parameters are simple, blittable types. Otherwise, P/Invoke performs very robust steps to convert each managed parameter to an appropriate native type, and vice versa if the arguments are marked as "out", or "in,out".

In other words, C++ Interop uses the fastest possible method of data marshaling, whereas P/Invoke uses the most robust method. This means that C++ Interop (in a fashion typical for C++) provides optimal performance by default, and the programmer is responsible for addressing cases where this behavior is not safe or appropriate.

C++ Interop therefore requires that data marshaling must be provided explicitly, but the advantage is that the programmer is free to decide what is appropriate, given the nature of the data, and how it is to be used. Furthermore, although the behavior of P/Invoke data marshaling can be modified at customized to a degree, C++ Interop allows data marshaling to be customized on a call-by-call basis. This is not possible with P/Invoke.

For more information about C++ Interop, see [Using C++ Interop \(Implicit PInvoke\)](#).

## See also

[Mixed \(Native and Managed\) Assemblies](#)

# Application domains and Visual C++

9/20/2022 • 2 minutes to read • [Edit Online](#)

If you have a `__clrcall` virtual function, the vtable will be per application domain (appdomain). If you create an object in one appdomain, you can only call the virtual function from within that appdomain. In mixed mode (`/clr`) you will have per-process vtables if your type has no `__clrcall` virtual functions. The `/clr:pure` and `/clr:safe` compiler options are deprecated in Visual Studio 2015 and unsupported in Visual Studio 2017.

For more information, see:

- [appdomain](#)
- [\\_\\_clrcall](#)
- [process](#)

## See also

- [Mixed \(Native and Managed\) Assemblies](#)

# Double Thunking (C++)

9/20/2022 • 3 minutes to read • [Edit Online](#)

Double thunking refers to the loss of performance you can experience when a function call in a managed context calls a Visual C++ managed function and where program execution calls the function's native entry point in order to call the managed function. This topic discusses where double thunking occurs and how you can avoid it to improve performance.

## Remarks

By default, when compiling with `/clr`, the definition of a managed function causes the compiler to generate a managed entry point and a native entry point. This allows the managed function to be called from native and managed call sites. However, when a native entry point exists, it can be the entry point for all calls to the function. If a calling function is managed, the native entry point will then call the managed entry point. In effect, two calls are required to invoke the function (hence, double thunking). For example, virtual functions are always called through a native entry point.

One resolution is to tell the compiler not to generate a native entry point for a managed function, that the function will only be called from a managed context, by using the `__clrcall` calling convention.

Similarly, if you export (`dllexport`, `dllimport`) a managed function, a native entry point is generated and any function that imports and calls that function will call through the native entry point. To avoid double thunking in this situation, do not use native export/import semantics; simply reference the metadata via `#using` (see [#using Directive](#)).

The compiler has been updated to reduce unnecessary double thunking. For example, any function with a managed type in the signature (including return type) will implicitly be marked as `__clrcall`.

## Example: Double thunking

### Description

The following sample demonstrates double thunking. When compiled native (without `/clr`), the call to the virtual function in `main` generates one call to `T`'s copy constructor and one call to the destructor. Similar behavior is achieved when the virtual function is declared with `/clr` and `__clrcall`. However, when just compiled with `/clr`, the function call generates a call to the copy constructor but there is another call to the copy constructor due to the native-to-managed thunk.

### Code

```

// double_thunking.cpp
// compile with: /clr
#include <stdio.h>
struct T {
    T() {
        puts(__FUNCSIG__);
    }

    T(const T&) {
        puts(__FUNCSIG__);
    }

    ~T() {
        puts(__FUNCSIG__);
    }

    T& operator=(const T&) {
        puts(__FUNCSIG__);
        return *this;
    }
};

struct S {
    virtual void /* __clrcall */ f(T t) {};
} s;

int main() {
    S* pS = &s;
    T t;

    printf("calling struct S\n");
    pS->f(t);
    printf("after calling struct S\n");
}

```

## Sample Output

```

__thiscall T::T(void)
calling struct S
__thiscall T::T(const struct T &)
__thiscall T::T(const struct T &)
__thiscall T::~T(void)
__thiscall T::~T(void)
after calling struct S
__thiscall T::~T(void)

```

## Example: Effect of double thunking

### Description

The previous sample demonstrated the existence of double thunking. This sample shows its effect. The `for` loop calls the virtual function and the program reports execution time. The slowest time is reported when the program is compiled with `/clr`. The fastest times are reported when compiling without `/clr` or if the virtual function is declared with `__clrcall`.

### Code

```
// double_thunking_2.cpp
// compile with: /clr
#include <time.h>
#include <stdio.h>

#pragma unmanaged
struct T {
    T() {}
    T(const T&) {}
    ~T() {}
    T& operator=(const T&) { return *this; }
};

struct S {
    virtual void /* __clrcall */ f(T t) {};
} s;

int main() {
    S* pS = &s;
    T t;
    clock_t start, finish;
    double duration;
    start = clock();

    for ( int i = 0 ; i < 1000000 ; i++ )
        pS->f(t);

    finish = clock();
    duration = (double)(finish - start) / (CLOCKS_PER_SEC);
    printf( "%2.1f seconds\n", duration );
    printf("after calling struct S\n");
}
```

## Sample Output

```
4.2 seconds
after calling struct S
```

## See also

[Mixed \(Native and Managed\) Assemblies](#)

# Avoiding Exceptions on CLR Shutdown When Consuming COM Objects Built with /clr

9/20/2022 • 2 minutes to read • [Edit Online](#)

Once the common language runtime (CLR) enters shutdown mode, native functions have limited access to CLR services. When attempting to call `Release` on a COM object compiled with `/clr`, the CLR transitions to native code and then transitions back into managed code to service the `IUnknown::Release` call (which is defined in managed code). The CLR prevents the call back into managed code since it is in shutdown mode.

To resolve this, ensure that destructors called from `Release` methods only contain native code.

## See also

[Mixed \(Native and Managed\) Assemblies](#)

# How to: Create a Partially Trusted Application by Removing Dependency on the CRT Library DLL

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic discusses how to create a partially trusted Common Language Runtime application using Visual C++ by removing dependency on msvc90.dll.

A Visual C++ application built with `/clr` will have a dependency on msvc90.dll, which is part of the C-Runtime Library. When you want your application to be used in a partial trust environment, the CLR will enforce certain code access security rules on your DLL. Therefore, it will be necessary to remove this dependency because msvc90.dll contains native code and code access security policy cannot be enforced on it.

If your application does not use any functionality of the C-Runtime Library and you would like to remove the dependency on this library from your code, you will have to use the `/NODEFAULTLIB:msvcrt.lib` linker option and link with either `ptrustm.lib` or `ptrustmd.lib`. These libraries contain object files for initialization and uninitialization of an application, exception classes used by the initialization code, and managed exception handling code. Linking in one of these libraries will remove any dependency on msvc90.dll.

## NOTE

The order of assembly uninitialization might differ for applications that use the ptrust libraries. For normal applications, assemblies are usually unloaded in the reverse order that they are loaded, but this is not guaranteed. For partial trust applications, assemblies are usually unloaded in the same order that they are loaded. This, also, is not guaranteed.

## To create a partially trusted mixed (/clr) application

1. To remove the dependency on msvc90.dll, you must specify to the linker not to include this library by using the `/NODEFAULTLIB:msvcrt.lib` linker option. For information on how to do this using the Visual Studio development environment or programmatically, see [/NODEFAULTLIB \(Ignore Libraries\)](#).
2. Add one of the `ptrustm` libraries to the linker input dependencies. Use `ptrustm.lib` if you are building your application in release mode. For debug mode, use `ptrustmd.lib`. For information on how to do this using the Visual Studio development environment or programmatically, see [.Lib Files as Linker Input](#).

## See also

[Mixed \(Native and Managed\) Assemblies](#)

[Initialization of Mixed Assemblies](#)

[Library Support for Mixed Assemblies](#)

[/link \(Pass Options to Linker\)](#)

# Using a Windows Form User Control in MFC

9/20/2022 • 2 minutes to read • [Edit Online](#)

Using the MFC Windows Forms support classes, you can host Windows Forms controls within your MFC applications as an ActiveX control within MFC dialog boxes or views. In addition, Windows Forms forms can be hosted as MFC dialog boxes.

The following sections describe how to:

- Host a Windows Forms control in an MFC dialog box.
- Host a Windows Forms user control as an MFC view.
- Host a Windows Forms form as an MFC dialog box.

## NOTE

MFC Windows Forms integration works only in projects that link dynamically with MFC (projects in which `_AFXDLL` is defined).

## NOTE

When you build your application using a private (modified) copy of the MFC Windows Forms interfaces DLL (`mfcmifc80.dll`), it will fail to install in the GAC unless you replace the Microsoft key with your own vendor key. For more information on assembly signing, see [Programming with Assemblies](#) and [Strong Name Assemblies \(Assembly Signing\) \(C++/CLI\)](#).

If your MFC application uses Windows Forms, you need to redistribute `mfcmifc80.dll` with your application. For more information, see [Redistributing the MFC Library](#).

## In This Section

[Hosting a Windows Form User Control in an MFC Dialog Box](#)

[Hosting a Windows Forms User Control as an MFC View](#)

[Hosting a Windows Form User Control as an MFC Dialog Box](#)

## Reference

[CWinFormsControl Class](#)

[CWinFormsDialog Class](#)

[CWinFormsView Class](#)

[ICommandSource Interface](#)

[ICommandTarget Interface](#)

[ICommandUI Interface](#)

[IView Interface](#)

[CommandHandler](#)

[DDX\\_ManagedControl](#)

[UICheckState](#)

## Related Sections

[Windows Forms](#)

[Windows Forms Controls](#)

## See also

[User Interface Elements](#)

[Form Views](#)

# Windows Forms/MFC Programming Differences

9/20/2022 • 2 minutes to read • [Edit Online](#)

The topics in [Using a Windows Form User Control in MFC](#) describe the MFC support for Windows Forms. If you are not familiar with .NET Framework or MFC programming, this topic provides background information about programming differences between the two.

Windows Forms is for creating Microsoft Windows applications on the .NET Framework. This framework provides a modern, object-oriented, extensible set of classes that enable you to develop rich Windows-based applications. With Windows Forms, you are able to create a rich client application that can access a wide variety of data sources and provide data-display and data-editing facilities using Windows Forms controls.

However, if you are accustomed to MFC, you might be used to creating certain types of applications that are not yet explicitly supported in Windows Forms. Windows Forms applications are equivalent to MFC dialog applications. However, they do not provide the infrastructure to directly support other MFC application types like OLE document server/container, ActiveX documents, the Document/View support for single-document interface (SDI), multiple-document interface (MDI), and multiple top-level interface (MTI). You can write your own logic to create these applications.

For more information about Windows Forms applications, see [Introduction to Windows Forms](#).

The following MFC view or document and command routing features have no equivalents in Windows Forms:

- Shell integration

MFC handles the dynamic data exchange (DDE) commands and command-line arguments that the shell uses when you right-click a document and select such verbs as Open, Edit, or Print. Windows Forms has no shell integration and does not respond to shell verbs.

- Document templates

In MFC, document templates associate a view, which is contained in a frame window (in MDI, SDI, or MTI mode), with the document you opened. Windows Forms has no equivalent to document templates.

- Documents

MFC registers document file types and processes the document type when opening a document from the shell. Windows Forms has no document support.

- Document states

MFC maintains dirty states for the document. Therefore, when you close the application, close the last view that contains the application, or exit from Windows, MFC prompts you to save the document. Windows Forms has no equivalent support.

- Commands

MFC has the concept of commands. The menu bar, toolbar, and context menu can all invoke the same command, for example, Cut and Copy. In Windows Forms, commands are tightly bound events from a particular UI element (such as a menu item); therefore, you have to hook up all the command events explicitly. You can also handle multiple events with a single handler in Windows Forms. For more information, see [Connecting Multiple Events to a Single Event Handler in Windows Forms](#).

- Command routing

MFC command routing enables the active view or document to process commands. Because the same command often has different meanings for different views (for example, Copy behaves differently in text edit view than in a graphics editor), the commands need to be handled by the active view. Because Windows Forms menus and toolbars have no inherent understanding of the active view, you cannot have a different handler for each view type for your `MenuItem.Click` events without writing additional internal code.

- Command update mechanism

MFC has a command update mechanism. Therefore, the active view or document is responsible for the state of the UI elements (for example, enabling or disabling a menu item or tool button, and checked states). Windows Forms has no equivalent of a command update mechanism.

## See also

[Using a Windows Form User Control in MFC](#)

# Hosting a Windows Form User Control in an MFC Dialog Box

9/20/2022 • 2 minutes to read • [Edit Online](#)

MFC hosts a Windows Forms control as a special kind of ActiveX control and communicates with the control by using ActiveX interfaces, and properties and methods of the [Control](#) class. We recommend that you use .NET Framework properties and methods to operate on the control.

## NOTE

In the current release, a `CDialogBar` object cannot host Windows Forms controls.

## In This Section

[How to: Create the User Control and Host in a Dialog Box](#)

[How to: Do DDX/DDV Data Binding with Windows Forms](#)

[How to: Sink Windows Forms Events from Native C++ Classes](#)

## Reference

[CWinFormsControl1](#) Class

[CDialog](#) Class

[CWnd](#) Class

[Control](#)

## See also

[Using a Windows Form User Control in MFC](#)

[Windows Forms/MFC Programming Differences](#)

[Hosting a Windows Forms User Control as an MFC View](#)

[Hosting a Windows Form User Control as an MFC Dialog Box](#)

# How to: Create the User Control and Host in a Dialog Box

9/20/2022 • 3 minutes to read • [Edit Online](#)

The steps in this article assume that you are creating a dialog-based ([CDialog Class](#)) Microsoft Foundation Classes (MFC) project, but you can also add support for a Windows Forms control to an existing MFC dialog box.

## To create the .NET user control

1. Create a Visual C# Windows Forms Control Library project named `WindowsFormsControlLibrary1`.

On the **File** menu, click **New** and then click **Project**. In the **Visual C#** folder, select **Windows Forms Control Library**.

Accept the `WindowsFormsControlLibrary1` project name by clicking **OK**.

By default, the name of the .NET control will be `UserControl1`.

2. Add child controls to `UserControl1`.

In the **Toolbox**, open the **All Windows Forms** list. Drag a **Button** control to the `UserControl1` design surface.

Also add a **TextBox** control.

3. In **Solution Explorer**, double-click `UserControl1.Designer.cs` to open it for editing. Change the declarations of the TextBox and the Button from `private` to `public`.

4. Build the project.

On the **Build** menu, click **Build Solution**.

## To create the MFC host application

1. Create an MFC Application project.

On the **File** menu, click **New** and then click **Project**. In the **Visual C++** folder, select **MFC Application**.

In the **Name** box, type `MFC01`. Change the Solution setting to **Add to Solution**. Click **OK**.

In the **MFC Application Wizard**, for Application Type, select **Dialog based**. Accept the remaining default settings and click **Finish**. This creates an MFC application that has an MFC dialog box.

2. Add a placeholder control to the MFC dialog box.

On the **View** menu, click **Resource View**. In **Resource View**, expand the **Dialog** folder and double-click `IDD_MFC01_DIALOG`. The dialog resource appears in **Resource Editor**.

In the **Toolbox**, open the **Dialog Editor** list. Drag a **Static Text** control to the dialog resource. The **Static Text** control will serve as a placeholder for the .NET Windows Forms control. Resize it to approximately the size of the Windows Forms control.

In the **Properties** window, change the ID of the **Static Text** control to `IDC_CTRL1` and change the **TabStop** property to **True**.

3. Configure the project for Common Language Runtime (CLR) support.

In **Solution Explorer**, right-click the `MFC01` project node, and then click **Properties**.

In the **Property Pages** dialog box, under **Configuration Properties**, select **General**. In the **Project Defaults** section, set **Common Language Runtime support** to **Common Language Runtime Support (/clr)**.

Under **Configuration Properties**, expand **C/C++** and select the **General** node. Set **Debug Information Format** to **Program Database (/Zi)**.

Select the **Code Generation** node. Set **Enable Minimal Rebuild** to **No (/Gm-)**. Also set **Basic Runtime Checks** to **Default**.

Click **OK** to apply the changes.

4. Add a reference to the .NET control.

In **Solution Explorer**, right-click the MFC01 project node and then click **Add, References**. On the **Property Page**, click **Add New Reference**, select **WindowsFormsControlLibrary1** (under the **Projects** tab), and click **OK**. This adds a reference in the form of a **/FU** compiler option so that the program will compile. It also puts a copy of WindowsFormsControlLibrary1.dll in the \MFC01\ project folder so that the program will run.

5. In Stdafx.h, find this line:

```
#endif // _AFX_NO_AFXCMN_SUPPORT
```

Above it, add these lines:

```
#include <afxwinforms.h> // MFC Windows Forms support
```

6. Add code to create the managed control.

First, declare the managed control. In MFC01Dlg.h, go to the declaration of the dialog class, and add a data member for the user control in Protected scope, as follows.

```
class CMFC01Dlg : public CDialog
{
    // ...
    // Data member for the .NET User Control:
    CWinFormsControl<WindowsFormsControlLibrary1::UserControl1> m_ctrl1;
```

Next, provide an implementation for the managed control. In MFC01Dlg.cpp, in the dialog override of **CMFC01Dlg::DoDataExchange** that was generated by the MFC Application wizard (not **CAboutDlg::DoDataExchange**, which is in the same file), add the following code to create the managed control and associate it with the static place holder IDC\_CTRL1.

```
void CMFC01Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_ManagedControl(pDX, IDC_CTRL1, m_ctrl1);
}
```

7. Build and run the project.

In **Solution Explorer**, right-click **MFC01** and then click **Set as StartUp Project**.

On the **Build** menu, click **Build Solution**.

On the **Debug** menu, click **Start without debugging**. The MFC dialog box should display the Windows

Forms control.

## See also

[Hosting a Windows Form User Control in an MFC Dialog Box](#)

# How to: Do DDX/DDV Data Binding with Windows Forms

9/20/2022 • 2 minutes to read • [Edit Online](#)

`DDX_ManagedControl` calls `CWinFormsControl::CreateManagedControl` to create a control matching the resource control ID. If you use `DDX_ManagedControl` for a `CWinFormsControl` control (in wizard-generated code), you should not call `CreateManagedControl` explicitly for the same control.

Call `DDX_ManagedControl` in `CWnd::DoDataExchange` to create controls from resource IDs. For data exchange, you do not need to use the DDX/DDV functions with Windows Forms controls. Instead, you can place code to access the properties of the managed control in the `DoDataExchange` method of your dialog (or view) class, as in the following example.

The following example shows how to bind a native C++ string to a .NET user control.

## Example: DDX/DDV data binding

The following is an example of DDX/DDV data binding of an MFC string `m_str` with the user-defined `NameText` property of a .NET user control.

The control is created when `CDialog::OnInitDialog` calls `CMyDlg::DoDataExchange` for the first time, so any code that references `m_UserControl1` must come after the `DDX_ManagedControl` call.

You can implement this code in the MFC01 application you created in [How to: Create the User Control and Host in a Dialog Box](#).

Put the following code in the declaration of CMFC01Dlg:

```
class CMFC01Dlg : public CDialog
{
    CWinFormsControl<WindowsFormsControlLibrary1::UserControl1> m_MyControl;
    CString m_str;
};
```

## Example: Implement DoDataExchange()

Put the following code in the implementation of CMFC01Dlg:

```
void CMFC01Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_ManagedControl(pDX, IDC_CTRL1, m_MyControl);

    if (pDX->m_bSaveAndValidate) {
        m_str = m_MyControl->textBox1->Text;
    } else {
        m_MyControl->textBox1->Text = gcnew System::String(m_str);
    }
}
```

## Example: Add handler method

Now we will add the handler method for a click on the OK button. Click the **Resource View** tab. In Resource View, double-click on **IDD\_MFC01\_DIALOG**. The dialog resource appears in Resource Editor. Then double click the OK button..

Define the handler as follows.

```
void CMFC01Dlg::OnBnClickedOk()
{
    AfxMessageBox(CString(m_MyControl.GetControl()->textBox1->Text));
    OnOK();
}
```

## Example: Set the textBox text

And add the following line to the implementation of **BOOL CMFC01Dlg::OnInitDialog()**.

```
m_MyControl.GetControl()->textBox1->Text = "hello";
```

You can now build and run the application. Notice that any text in the text box will be displayed in a pop-up message box when the application closes.

## See also

[CWinFormsControl Class](#)

[DDX\\_ManagedControl](#)

[CWnd::DoDataExchange](#)

# How to: Sink Windows Forms Events from Native C++ Classes

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can enable native C++ classes to receive callbacks from managed events raised from Windows Forms controls or other forms with the MFC macro map format. Sinking events in views and dialogs is similar to doing the same task for controls.

To do this, you need to:

- Attach an `OnClick` event handler to the control using [MAKE\\_DELEGATE](#).
- Create a delegate map using [BEGIN\\_DELEGATE\\_MAP](#), [END\\_DELEGATE\\_MAP](#), and [EVENT\\_DELEGATE\\_ENTRY](#).

This sample continues the work you did in [How to: Do DDX/DDV Data Binding with Windows Forms](#).

Now, you will associate your MFC control (`m_MyControl`) with a managed event handler delegate called `onClick` for the managed `Click` event.

## To attach the OnClick event handler:

1. Add the following code to the implementation of `BOOL CMFC01Dlg::OnInitDialog`:

```
m_MyControl.GetControl()->button1->Click += MAKE_DELEGATE( System::EventHandler, OnClick );
```

2. Add the following code to the public section in the declaration of class `CMFC01Dlg : public CDialog`.

```
// delegate map
BEGIN_DELEGATE_MAP( CMFC01Dlg )
EVENT_DELEGATE_ENTRY( OnClick, System::Object^, System::EventArgs^ )
END_DELEGATE_MAP()

void OnClick( System::Object^ sender, System::EventArgs^ e );
```

3. Finally, add the implementation for `onClick` to `CMFC01Dlg.cpp`:

```
void CMFC01Dlg::OnClick(System::Object^ sender, System::EventArgs^ e)
{
    AfxMessageBox(_T("Button clicked"));
}
```

## See also

[MAKE\\_DELEGATE](#)  
[BEGIN\\_DELEGATE\\_MAP](#)  
[END\\_DELEGATE\\_MAP](#)  
[EVENT\\_DELEGATE\\_ENTRY](#)

# Hosting a Windows Forms User Control as an MFC View

9/20/2022 • 2 minutes to read • [Edit Online](#)

MFC uses the `CWinFormsView` class to host a Windows Forms user control in an MFC view. MFC Windows Forms views are ActiveX controls. The user control is hosted as a child of the native view and occupies the entire client area of the native view.

The end result resembles the model used by the [CFormView Class](#). This lets you take advantage of the Windows Forms designer and runtime to create rich form-based views.

Because MFC Windows Forms views are ActiveX controls, they do not have the same `hwnd` as MFC views. Also they cannot be passed as a pointer to a `CView` view. In general, use .NET Framework methods to work with Windows Forms views and rely less on Win32.

## In This Section

[How to: Create the User Control and Host MDI View](#)

[How to: Add Command Routing to the Windows Forms Control](#)

[How to: Call Properties and Methods of the Windows Forms Control](#)

## See also

[Using a Windows Form User Control in MFC](#)

[How to: Author Composite Controls](#)

# How to: Create the User Control and Host MDI View

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following steps show how to create a .NET Framework user control, author the user control in a control class library (specifically, a Windows Control Library project), and then compile the project into an assembly. The control can then be consumed from an MFC application that uses classes derived from [CView Class](#) and [CWinFormsView Class](#).

For information about how to create a Windows Forms user control and author a control class library, see [How to: Author User Controls](#).

## NOTE

In some cases, Windows Forms controls, such as a third-party Grid control, might not behave reliably when hosted in an MFC application. A recommended workaround is to place a Windows Forms User Control in the MFC application and place the third-party Grid control inside the User control.

This procedure assumes that you created a Windows Forms Controls Library project named WindowsFormsControlLibrary1, as per the procedure in [How to: Create the User Control and Host in a Dialog Box](#).

## To create the MFC host application

1. Create an MFC Application project.

On the **File** menu, select **New**, and then click **Project**. In the **Visual C++** folder, select **MFC Application**.

In the **Name** box, enter **MFC02** and change the **Solution** setting to **Add to Solution**. Click **OK**.

In the **MFC Application Wizard**, accept all the defaults, and then click **Finish**. This creates an MFC application with a Multiple Document Interface.

2. Configure the project for Common Language Runtime (CLR) support.

In **Solution Explorer**, right-click the **MFC01** project node, and select **Properties** from the context menu. The **Property Pages** dialog box appears.

Under **Configuration Properties**, select **General**. Under the **Project Defaults** section, set **Common Language Runtime support** to **Common Language Runtime Support (/clr)**.

Under **Configuration Properties**, expand **C/C++** and click the **General** node. Set **Debug Information Format** to **Program Database (/Zi)**.

Click the **Code Generation** node. Set **Enable Minimal Rebuild** to **No (/Gm-)**. Also set **Basic Runtime Checks** to **Default**.

Click **OK** to apply your changes.

3. In *pch.h* (*stdafx.h* in Visual Studio 2017 and earlier), add the following line:

```
#using <System.Windows.Forms.dll>
```

4. Add a reference to the .NET control.

In Solution Explorer, right-click the `MFC02` project node and select **Add, References**. In the **Property Page**, click **Add New Reference**, select `WindowsFormsControlLibrary1` (under the **Projects** tab), and click **OK**. This adds a reference in the form of a `/FU` compiler option so that the program will compile; it also copies `WindowsFormsControlLibrary1.dll` into the `MFC02` project directory so that the program will run.

5. In `stdafx.h`, find this line:

```
#endif // _AFX_NO_AFXCMN_SUPPORT
```

Add these lines above it:

```
#include <afxwinforms.h> // MFC Windows Forms support
```

6. Modify the view class so that it inherits from `CWinFormsView`.

In `MFC02View.h`, replace `CView` with `CWinFormsView` so that the code appears as follows:

```
class CMFC02View : public CWinFormsView
{
};
```

If you want add additional views to your MDI application, you will need to call `CWinApp::AddDocTemplate` for each view you create.

7. Modify the `MFC02View.cpp` file to change `CView` to `CWinFormsView` in the `IMPLEMENT_DYNCREATE` macro and message map and replace the existing empty constructor with the constructor shown below:

```
IMPLEMENT_DYNCREATE(CMFC02View, CWinFormsView)

CMFC02View::CMFC02View(): CWinFormsView(WindowsFormsControlLibrary1::UserControl1::typeid)
{
}
BEGIN_MESSAGE_MAP(CMFC02View, CWinFormsView)
//leave existing body as is
END_MESSAGE_MAP()
```

8. Build and run the project.

In Solution Explorer, right-click `MFC02` and select **Set as StartUp Project**.

On the **Build** menu, click **Build Solution**.

On the **Debug** menu, click **Start without debugging**.

## See also

[Hosting a Windows Forms User Control as an MFC View](#)

# How to: Add Command Routing to the Windows Forms Control

9/20/2022 • 2 minutes to read • [Edit Online](#)

`CWinFormsView` routes commands and update-command UI messages to the user control to allow it to handle MFC commands (for example, frame menu items and toolbar buttons).

The user control uses `ICommandTarget::Initialize` to store a reference to the command source object in `m_CmdSrc`, as shown in the following example. To use `ICommandTarget` you must add a reference to `mfcmifc80.dll`.

`CWinFormsView` handles several of the common MFC view notifications by forwarding them to the managed user control. These notifications include the `OnInitialUpdate`, `OnUpdate` and `OnActivateView` methods.

This topic assumes you have previously completed [How to: Create the User Control and Host in a Dialog Box](#) and [How to: Create the User Control and Host MDI View](#).

## To create the MFC host application

1. Open Windows Forms Control Library you created in [How to: Create the User Control and Host in a Dialog Box](#).
2. Add a reference to `mfcmifc80.dll`, which you can do by right-clicking the project node in **Solution Explorer**, selecting **Add, Reference**, and then browsing to Microsoft Visual Studio 10.0\VC\atlmfc\lib.
3. Open `UserControl1.Designer.cs` and add the following using statement:

```
using Microsoft.VisualC.MFC;
```

4. Also, in `UserControl1.Designer.cs`, change this line:

```
partial class UserControl1
```

to this:

```
partial class UserControl1 : System.Windows.Forms.UserControl, ICommandTarget
```

5. Add this as the first line of the class definition for `UserControl1`:

```
private ICommandSource m_CmdSrc;
```

6. Add the following method definitions to `UserControl1` (we will create the ID of the MFC control in the next step):

```
public void Initialize (ICommandSource cmdSrc)
{
    m_CmdSrc = cmdSrc;
    // need ID of control in MFC dialog and callback function
    m_CmdSrc.AddCommandHandler(32771, new CommandHandler (singleMenuHandler));
}

private void singleMenuHandler (uint cmdUI)
{
    // User command handler code
    System.Windows.Forms.MessageBox.Show("Custom menu option was clicked.");
}
```

7. Open the MFC application you created in [How to: Create the User Control and Host MDI View](#).

8. Add a menu option that will invoke `singleMenuHandler`.

Go to **Resource View** (Ctrl+Shift+E), expand the **Menu** folder, and then double-click **IDR\_MFC02TYPE**. This displays the menu editor.

Add a menu option at the bottom of the **View** menu. Notice the ID of the menu option in the **Properties** window. Save the file.

In **Solution Explorer**, open the Resource.h file, copy the ID value for the menu option you just added, and paste that value as the first parameter to the `m_CmdSrc.AddCommandHandler` call in the C# project's `Initialize` method (replacing `32771` if necessary).

9. Build and run the project.

On the **Build** menu, click **Build Solution**.

On the **Debug** menu, click **Start without debugging**.

Select the menu option you added. Notice that the method in the .dll is called.

## See also

[Hosting a Windows Forms User Control as an MFC View](#)

[ICommandSource Interface](#)

[ICommandTarget Interface](#)

# How to: Call Properties and Methods of the Windows Forms Control

9/20/2022 • 2 minutes to read • [Edit Online](#)

Because `CWinFormsView::GetControl` returns a pointer to `System.Windows.Forms.Control`, and not a pointer to `WindowsControlLibrary1::UserControl1`, it is advisable to add a member of the user control type and initialize it in `IView::OnInitialUpdate`. Now you can call methods and properties using `m_ViewControl`.

This topic assumes you have previously completed [How to: Create the User Control and Host in a Dialog Box](#) and [How to: Create the User Control and Host MDI View](#).

## To create the MFC host application

1. Open the MFC application you created in [How to: Create the User Control and Host MDI View](#).
2. Add the following line to the public overrides section of the `CMFC02View` class declaration in `MFC02View.h`.

```
gcroot<WindowsFormsControlLibrary1::UserControl1 ^> m_ViewControl;
```

3. Add an override for `OnInitialupdate`.

Display the **Properties** window (F4). In **Class View** (CTRL+SHIFT+C), select `CMFC02View` class. In the **Properties** window, select the icon for Overrides. Scroll down the list to `OnInitialUpdate`. Click on the drop down list and select <Add>. In `MFC02View.cpp`, make sure the body of the `OnInitialUpdate` function is as follows:

```
CWinFormsView::OnInitialUpdate();
m_ViewControl = safe_cast<WindowsFormsControlLibrary1::UserControl1 ^>(this->GetControl());
m_ViewControl->textBox1->Text = gcnew System::String("hi");
```

4. Build and run the project.

On the **Build** menu, click **Build Solution**.

On the **Debug** menu, click **Start without debugging**.

Notice that the text box is now initialized.

## See also

[Hosting a Windows Forms User Control as an MFC View](#)

# Hosting a Windows Form User Control as an MFC Dialog Box

9/20/2022 • 3 minutes to read • [Edit Online](#)

MFC provides the template class `CWinFormsDialog` so that you can host a Windows Forms user control (`UserControl`) in a modal or modeless MFC dialog box. `CWinFormsDialog` is derived from the MFC class `CDialog`, so the dialog box can be launched as modal or modeless.

The process that `CWinFormsDialog` uses to host the user control is similar to that described in [Hosting a Windows Form User Control in an MFC Dialog Box](#). However, `CWinFormsDialog` manages the initialization and hosting of the user control so that it does not have to be programmed manually.

## To create the MFC host application

1. Create an MFC Application project.

On the **File** menu, select **New**, and then click **Project**. In the **Visual C++** folder, select **MFC Application**.

In the **Name** box, enter `MFC03` and change the Solution setting to **Add to Solution**. Click **OK**.

In the **MFC Application Wizard**, accept all the defaults, and then click **Finish**. This creates an MFC application with a Multiple Document Interface.

2. Configure the project.

In **Solution Explorer**, right-click the **MFC03** project node, and choose **Properties**. The **Property Pages** dialog box appears.

In the **Property Pages** dialog box, select **Configuration Properties > General**. In the **Project Defaults** section, set **Common Language Runtime support** to **Common Language Runtime Support (/clr)**. Choose **OK**.

3. Add a reference to the .NET control.

In **Solution Explorer**, right-click the **MFC03** project node and choose **Add, References**. In the **Property Page**, click **Add New Reference**, select **WindowsControlLibrary1** (under the **Projects** tab), and click **OK**. This adds a reference in the form of a `/FU` compiler option so that the program will compile; it also copies **WindowsControlLibrary1.dll** into the `MFC03` project directory so that the program will run.

4. Add `#include <afxwinforms.h>` to `pch.h` (`stdafx.h` in Visual Studio 2017 and earlier), at the end of the existing `#include` statements.

5. Add a new class that subclasses `CDialog`.

Right click on project name and add an MFC class (called **CHostForWinForm**) that subclasses `CDialog`. Since you do not need the dialog box resource, you can delete the resource ID (select **Resource View**, expand the **Dialog** folder and delete `IDD_HOSTFORWINFORM` resource. Then, remove any references to the ID in code.).

6. Replace `CDialog` in **CHostForWinForm.h** and **CHostForWinForm.cpp** files with `CWinFormsDialog<WindowsControlLibrary1::UserControl1>`.

7. Call `DoModal` on the **CHostForWinForm** class.

In MFC03.cpp, add `#include "HostForWinForm.h"`.

Before the return statement in the definition of CMFC03App::InitInstance, add:

```
CHostForWinForm m_HostForWinForm;
m_HostForWinForm.DoModal();
```

8. Build and run the project.

On the **Build** menu, click **Build Solution**.

On the **Debug** menu, click **Start without debugging**.

Next you will add code to monitor the state of a control on the Windows Forms from the MFC application.

9. Add a handler for OnInitDialog.

Display the **Properties** window (F4). In **Class View**, select CHostForWinForm. In the **Properties** window, select overrides and in the row for OnInitDialog, click in the left hand column and select < Add >. This adds the following line to CHostForWinForm.h:

```
virtual BOOL OnInitDialog();
```

10. Define OnInitDialog (in CHostForWinForm.cpp) as follows:

```
BOOL CHostForWinForm::OnInitDialog() {
    CWinFormsDialog<WindowsControlLibrary1::UserControl1>::OnInitDialog();
    GetControl()->button1->Click += MAKE_DELEGATE(System::EventHandler, OnButton1);
    return TRUE;
}
```

11. Next add the OnButton1 handler. Add the following lines to the public section of the CHostForWinForm class in CHostForWinForm.h:

```
virtual void OnButton1( System::Object^ sender, System::EventArgs^ e );
BEGIN_DELEGATE_MAP( CHostForWinForm )
    EVENT_DELEGATE_ENTRY( OnButton1, System::Object^, System::EventArgs^ );
END_DELEGATE_MAP()
```

In CHostForWinForm.cpp, add this definition:

```
void CHostForWinForm::OnButton1( System::Object^ sender, System::EventArgs^ e )
{
    System::Windows::Forms::MessageBox::Show("test");
}
```

12. Build and run the project. When you click the button, which is on the Windows Form, code in the MFC application will run.

Next you will add code to display from the MFC code the value in the text box on the Windows Form.

13. In the public section of the CHostForWinForm class in CHostForWinForm.h, add the following declaration:

```
CString m_sEditBoxOnWinForm;
```

14. In the definition of DoDataExchange in CHostForWinForm.cpp, add the following three lines to the end of the function:

```
if (pDX->m_bSaveAndValidate)
    m_sEditBoxOnWinForm = CString( GetControl()->textBox1->Text);
else
    GetControl()->textBox1->Text = gcnew System::String(m_sEditBoxOnWinForm);
```

15. In the definition of OnButton1 in CHostForWinForm.cpp, add the following three lines to the end of the function:

```
this->UpdateData(TRUE);
System::String ^ z = gcnew System::String(m_sEditBoxOnWinForm);
System::Windows::Forms::MessageBox::Show(z);
```

16. Build and run the project.

## See also

[System.Windows.Forms.UserControl Using a Windows Form User Control in MFC](#)

# Calling Native Functions from Managed Code

9/20/2022 • 7 minutes to read • [Edit Online](#)

The common language runtime provides Platform Invocation Services, or PInvoke, that enables managed code to call C-style functions in native dynamic-linked libraries (DLLs). The same data marshaling is used as for COM interoperability with the runtime and for the "It Just Works," or IJW, mechanism.

For more information, see:

- [Using Explicit PInvoke in C++ \(DllImport Attribute\)](#)
- [Using C++ Interop \(Implicit PInvoke\)](#)

The samples in this section just illustrate how `PInvoke` can be used. `PInvoke` can simplify customized data marshaling because you provide marshaling information declaratively in attributes instead of writing procedural marshaling code.

## NOTE

The marshaling library provides an alternative way to marshal data between native and managed environments in an optimized way. See [Overview of Marshaling in C++](#) for more information about the marshaling library. The marshaling library is usable for data only, and not for functions.

## PInvoke and the DllImport Attribute

The following example shows the use of `PInvoke` in a Visual C++ program. The native function `puts` is defined in `msvcrt.dll`. The `DllImport` attribute is used for the declaration of `puts`.

```
// platform_invocation_services.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[DllImport("msvcrt", CharSet=CharSet::Ansi)]
extern "C" int puts(String ^);

int main() {
    String ^ pStr = "Hello World!";
    puts(pStr);
}
```

The following sample is equivalent to the previous sample, but uses IJW.

```

// platform_invocation_services_2.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

#include <stdio.h>

int main() {
    String ^ pStr = "Hello World!";
    char* pChars = (char*)Marshal::StringToHGlobalAnsi(pStr).ToPointer();
    puts(pChars);

    Marshal::FreeHGlobal((IntPtr)pChars);
}

```

### Advantages of IJW

- There is no need to write `DLLImport` attribute declarations for the unmanaged APIs the program uses. Just include the header file and link with the import library.
- The IJW mechanism is slightly faster (for example, the IJW stubs do not need to check for the need to pin or copy data items because that is done explicitly by the developer).
- It clearly illustrates performance issues. In this case, the fact that you are translating from a Unicode string to an ANSI string and that you have an attendant memory allocation and deallocation. In this case, a developer writing the code using IJW would realize that calling `_putws` and using `PtrToStringChars` would be better for performance.
- If you call many unmanaged APIs using the same data, marshaling it once and passing the marshaled copy is much more efficient than re-marshaling every time.

### Disadvantages of IJW

- Marshaling must be specified explicitly in code instead of by attributes (which often have appropriate defaults).
- The marshaling code is inline, where it is more invasive in the flow of the application logic.
- Because the explicit marshaling APIs return `IntPtr` types for 32-bit to 64-bit portability, you must use extra `ToPointer` calls.

The specific method exposed by C++ is the more efficient, explicit method, at the cost of some additional complexity.

If the application uses mainly unmanaged data types or if it calls more unmanaged APIs than .NET Framework APIs, we recommend that you use the IJW feature. To call an occasional unmanaged API in a mostly managed application, the choice is more subtle.

## PInvoke with Windows APIs

PInvoke is convenient for calling functions in Windows.

In this example, a Visual C++ program interoperates with the `MessageBox` function that is part of the Win32 API.

```

// platform_invocation_services_4.cpp
// compile with: /clr /c
using namespace System;
using namespace System::Runtime::InteropServices;
typedef void* HWND;
[DllImport("user32", CharSet=CharSet::Ansi)]
extern "C" int MessageBox(HWND hWnd, String ^ pText, String ^ pCaption, unsigned int uType);

int main() {
    String ^ pText = "Hello World! ";
    String ^ pCaption = "PInvoke Test";
    MessageBox(0, pText, pCaption, 0);
}

```

The output is a message box that has the title PInvoke Test and contains the text Hello World!.

The marshaling information is also used by PInvoke to look up functions in the DLL. In user32.dll there is in fact no MessageBox function, but CharSet=CharSet::Ansi enables PInvoke to use MessageBoxA, the ANSI version, instead of MessageBoxW, which is the Unicode version. In general, we recommend that you use Unicode versions of unmanaged APIs because that eliminates the translation overhead from the native Unicode format of .NET Framework string objects to ANSI.

## When Not to Use PInvoke

Using PInvoke is not appropriate for all C-style functions in DLLs. For example, suppose there is a function MakeSpecial in mylib.dll declared as follows:

```
char * MakeSpecial(char * pszString);
```

If we use PInvoke in a Visual C++ application, we might write something similar to the following:

```

[DllImport("mylib")]
extern "C" String * MakeSpecial([MarshalAs(UnmanagedType::LPStr)] String ^);

```

The difficulty here is that we cannot delete the memory for the unmanaged string returned by MakeSpecial. Other functions called through PInvoke return a pointer to an internal buffer that does not have to be deallocated by the user. In this case, using the IJW feature is the obvious choice.

## Limitations of PInvoke

You cannot return the same exact pointer from a native function that you took as a parameter. If a native function returns the pointer that has been marshaled to it by PInvoke, memory corruption and exceptions may ensue.

```

__declspec(dllexport)
char* fstringA(char* param) {
    return param;
}

```

The following sample exhibits this problem, and even though the program may seem to give the correct output, the output is coming from memory that had been freed.

```

// platform_invocation_services_5.cpp
// compile with: /clr /c
using namespace System;
using namespace System::Runtime::InteropServices;
#include <limits.h>

ref struct MyPInvokeWrap {
public:
    [DllImport("user32.dll", EntryPoint = "CharLower", CharSet = CharSet::Ansi) ]
    static String^ CharLower([In, Out] String ^);
};

int main() {
    String ^ strout = "AabCc";
    Console::WriteLine(strout);
    strout = MyPInvokeWrap::CharLower(strout);
    Console::WriteLine(strout);
}

```

## Marshaling Arguments

With `PInvoke`, no marshaling is needed between managed and C++ native primitive types with the same form. For example, no marshaling is required between `Int32` and `int`, or between `Double` and `double`.

However, you must marshal types that do not have the same form. This includes `char`, `string`, and `struct` types. The following table shows the mappings used by the marshaler for various types:

WTYPES.H	VISUAL C++	VISUAL C++ WITH /CLR	COMMON LANGUAGE RUNTIME
HANDLE	void *	void *	IntPtr, UIntPtr
BYTE	unsigned char	unsigned char	Byte
SHORT	short	short	Int16
WORD	unsigned short	unsigned short	UInt16
INT	int	int	Int32
UINT	unsigned int	unsigned int	UInt32
LONG	long	long	Int32
BOOL	long	bool	Boolean
DWORD	unsigned long	unsigned long	UInt32
ULONG	unsigned long	unsigned long	UInt32
CHAR	char	char	Char
LPSTR	char *	String ^ [in], StringBuilder ^ [in, out]	String ^ [in], StringBuilder ^ [in, out]
LPCSTR	const char *	String ^	String

WTYPES.H	VISUAL C++	VISUAL C++ WITH /CLR	COMMON LANGUAGE RUNTIME
LPWSTR	wchar_t *	String ^ [in], StringBuilder ^ [in, out]	String ^ [in], StringBuilder ^ [in, out]
LPCWSTR	const wchar_t *	String ^	String
FLOAT	float	float	Single
DOUBLE	double	double	Double

The marshaler automatically pins memory allocated on the runtime heap if its address is passed to an unmanaged function. Pinning prevents the garbage collector from moving the allocated block of memory during compaction.

In the example shown earlier in this topic, the CharSet parameter of `DllImport` specifies how managed Strings should be marshaled; in this case, they should be marshaled to ANSI strings for the native side.

You can specify marshaling information for individual arguments of a native function by using the `MarshalAs` attribute. There are several choices for marshaling a `String ^` argument: `BStr`, `ANSIBStr`, `TBStr`, `LPStr`, `LPWStr`, and `LPTStr`. The default is `LPStr`.

In this example, the string is marshaled as a double-byte Unicode character string, `LPWStr`. The output is the first letter of Hello World! because the second byte of the marshaled string is null, and puts interprets this as the end-of-string marker.

```
// platform_invocation_services_3.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[DllImport("msvcrt", EntryPoint="puts")]
extern "C" int puts([MarshalAs(UnmanagedType::LPWStr)] String ^);

int main() {
    String ^ pStr = "Hello World!";
    puts(pStr);
}
```

The `MarshalAs` attribute is in the `System::Runtime::InteropServices` namespace. The attribute can be used with other data types such as arrays.

As mentioned earlier in the topic, the marshaling library provides a new, optimized method of marshaling data between native and managed environments. For more information, see [Overview of Marshaling in C++](#).

## Performance Considerations

PInvoke has an overhead of between 10 and 30 x86 instructions per call. In addition to this fixed cost, marshaling creates additional overhead. There is no marshaling cost between blittable types that have the same representation in managed and unmanaged code. For example, there is no cost to translate between `int` and `Int32`.

For better performance, have fewer PInvoke calls that marshal as much data as possible, instead of more calls that marshal less data per call.

## See also



# Using Explicit PInvoke in C++ (DllImport Attribute)

9/20/2022 • 2 minutes to read • [Edit Online](#)

The .NET Framework provides explicit Platform Invoke (or PInvoke) features with the `DllImport` attribute to allow managed applications to call unmanaged functions packaged inside DLLs. Explicit PInvoke is required for situations where unmanaged APIs are packaged as DLLs and the source code is not available. Calling Win32 functions, for example, requires PInvoke. Otherwise, use implicit PInvoke; see [Using C++ Interop \(Implicit PInvoke\)](#) for more information.

PInvoke works by using [DllImportAttribute](#). This attribute, which takes the name of the DLL as its first argument, is placed before a function declaration for each DLL entry point that will be used. The signature of the function must match the name of a function exported by the DLL (but some type conversion can be performed implicitly by defining the `DllImport` declarations in terms of managed types.)

The result is a managed entry point for each native DLL function that contains the necessary transition code (or thunk) and simple data conversions. Managed functions can then call into the DLL through these entry points. The code inserted into a module as the result of PInvoke is entirely managed.

## In This Section

- [Calling Native Functions from Managed Code](#)
- [How to: Call Native DLLs from Managed Code Using PInvoke](#)
- [How to: Marshal Strings Using PInvoke](#)
- [How to: Marshal Structures Using PInvoke](#)
- [How to: Marshal Arrays Using PInvoke](#)
- [How to: Marshal Function Pointers Using PInvoke](#)
- [How to: Marshal Embedded Pointers Using PInvoke](#)

## See also

[Calling Native Functions from Managed Code](#)

# How to: Call Native DLLs from Managed Code Using PInvoke

9/20/2022 • 2 minutes to read • [Edit Online](#)

Functions that are implemented in unmanaged DLLs can be called from managed code using Platform Invoke (P/Invoke) functionality. If the source code for the DLL is not available, P/Invoke is the only option for interoperating. However, unlike other .NET languages, Visual C++ provides an alternative to P/Invoke. For more information, see [Using C++ Interop \(Implicit PInvoke\)](#).

## Example

The following code example uses the Win32 [GetSystemMetrics](#) function to retrieve the current resolution of the screen in pixels.

For functions that use only intrinsic types as arguments and return values, no extra work is required. Other data types, such as function pointers, arrays, and structures, require additional attributes to ensure proper data marshaling.

Although it is not required, it is good practice to make P/Invoke declarations static members of a value class so that they do not exist in the global namespace, as demonstrated in this example.

```
// pinvoke_basic.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

value class Win32 {
public:
    [DllImport("User32.dll")]
    static int GetSystemMetrics(int);

    enum class SystemMetricIndex {
        // Same values as those defined in winuser.h.
        SM_CXSCREEN = 0,
        SM_CYSCREEN = 1
    };
};

int main() {
    int hRes = Win32::GetSystemMetrics( safe_cast<int>(Win32::SystemMetricIndex::SM_CXSCREEN) );
    int vRes = Win32::GetSystemMetrics( safe_cast<int>(Win32::SystemMetricIndex::SM_CYSCREEN) );
    Console::WriteLine("screen resolution: {0},{1}", hRes, vRes);
}
```

## See also

[Using Explicit PInvoke in C++ \(DllImport Attribute\)](#)

# How to: Marshal strings using P/Invoke

9/20/2022 • 3 minutes to read • [Edit Online](#)

Native functions that accept C-style strings can be called using the CLR string type `System::String` by using .NET Framework Platform Invoke (P/Invoke) support. We encourage you to use the C++ Interop features instead of P/Invoke when possible, because P/Invoke provides little compile-time error reporting, isn't type-safe, and can be tedious to implement. If the unmanaged API is packaged as a DLL, and the source code isn't available, then P/Invoke is the only option. Otherwise, see [Using C++ Interop \(Implicit P/Invoke\)](#).

Managed and unmanaged strings are laid out differently in memory, so passing strings from managed to unmanaged functions requires the [MarshalAsAttribute](#) attribute to instruct the compiler to insert the required conversion mechanisms for marshaling the string data correctly and safely.

As with functions that use only intrinsic data types, [DllImportAttribute](#) is used to declare managed entry points into the native functions. Functions that pass strings can use a handle to the `String` type instead of defining these entry points as taking C-style strings. Using this type prompts the compiler to insert code that performs the required conversion. For each function argument in an unmanaged function that takes a string, use the [MarshalAsAttribute](#) attribute to indicate that the `String` object should be marshaled to the native function as a C-style string.

The marshaler wraps the call to the unmanaged function in a hidden wrapper routine. The wrapper routine pins and copies the managed string into a locally allocated string in the unmanaged context. The local copy is then passed to the unmanaged function. When the unmanaged function returns, the wrapper deletes the resource. Or, if it was on the stack, it's reclaimed when the wrapper goes out of scope. The unmanaged function isn't responsible for this memory. The unmanaged code only creates and deletes memory in the heap set up by its own CRT, so there's never an issue with the marshaller using a different CRT version.

If your unmanaged function returns a string, either as a return value or an out parameter, the marshaler copies it into a new managed string, and then releases the memory. For more information, see [Default Marshaling Behavior](#) and [Marshaling Data with Platform Invoke](#).

## Example

The following code consists of an unmanaged module and a managed module. The unmanaged module is a DLL that defines a function called `TakesAString`. `TakesAString` accepts a C-style narrow string in the form of a `char*`.

```

// TraditionalDLL2.cpp
// compile with: /LD /EHsc
#include <windows.h>
#include <stdio.h>
#include <iostream>

using namespace std;

#define TRADITIONALDLL_EXPORTS
#ifdef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

extern "C" {
    TRADITIONALDLL_API void TakesAString(char*);
}

void TakesAString(char* p) {
    printf_s("[unmanaged] %s\n", p);
}

```

The managed module is a command-line application that imports the `TakesAString` function, but defines it as taking a managed `System.String` instead of a `char*`. The `MarshalAsAttribute` attribute is used to indicate how the managed string should be marshaled when `TakesAString` is called.

```

// MarshalString.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

value struct TraditionalDLL
{
    [DllImport("TraditionalDLL2.dll")]
    static public void
    TakesAString([MarshalAs(UnmanagedType::LPStr)]String^);
};

int main() {
    String^ s = gcnew String("sample string");
    Console::WriteLine("[managed] passing managed string to unmanaged function...");
    TraditionalDLL::TakesAString(s);
    Console::WriteLine("[managed] {0}", s);
}

```

This technique constructs a copy of the string on the unmanaged heap, so changes made to the string by the native function won't be reflected in the managed copy of the string.

No portion of the DLL is exposed to the managed code by the traditional `#include` directive. In fact, the DLL is accessed at runtime only, so problems in functions imported by using `DllImport` aren't detected at compile time.

## See also

[Using explicit P/Invoke in C++ \(`DllImport` attribute\)](#)

# How to: Marshal Structures Using P/Invoke

9/20/2022 • 4 minutes to read • [Edit Online](#)

This document explains how native functions that accept C-style structs can be called from managed functions by using P/Invoke. Although we recommend that you use the C++ Interop features instead of P/Invoke because P/Invoke provides little compile-time error reporting, is not type-safe, and can be tedious to implement, if the unmanaged API is packaged as a DLL and the source code is not available, P/Invoke is the only option.

Otherwise, see the following documents:

- [Using C++ Interop \(Implicit P/Invoke\)](#)
- [How to: Marshal Strings Using P/Invoke](#)

By default, native and managed structures are laid out differently in memory, so successfully passing structures across the managed/unmanaged boundary requires extra steps to preserve data integrity.

This document explains the steps required to define managed equivalents of native structures and how the resulting structures can be passed to unmanaged functions. This document assumes that simple structures — those that do not contain strings or pointers — are used. For information about non-blittable interoperability, see [Using C++ Interop \(Implicit P/Invoke\)](#). P/Invoke cannot have non-blittable types as a return value. Blittable types have the same representation in managed and unmanaged code. For more information, see [Blittable and Non-Blittable Types](#).

Marshaling simple, blittable structures across the managed/unmanaged boundary first requires that managed versions of each native structure be defined. These structures can have any legal name; there is no relationship between the native and managed version of the two structures other than their data layout. Therefore, it is vital that the managed version contains fields that are the same size and in the same order as the native version. (There is no mechanism for ensuring that the managed and native versions of the structure are equivalent, so incompatibilities will not become apparent until run time. It is the programmer's responsibility to ensure that the two structures have the same data layout.)

Because the members of managed structures are sometimes rearranged for performance purposes, it is necessary to use the [StructLayoutAttribute](#) attribute to indicate that the structure are laid out sequentially. It is also a good idea to explicitly set the structure packing setting to be the same as that used by the native structure. (Although by default, Visual C++ uses an 8-byte structure packing for both managed code.)

1. Next, use [DllImportAttribute](#) to declare entry points that correspond to any unmanaged functions that accept the structure, but use the managed version of the structure in the function signatures, which is a moot point if you use the same name for both versions of the structure.
2. Now managed code can pass the managed version of the structure to the unmanaged functions as though they are actually managed functions. These structures can be passed either by value or by reference, as demonstrated in the following example.

## Unmanaged and a managed modules

The following code consists of an unmanaged and a managed module. The unmanaged module is a DLL that defines a structure called Location and a function called GetDistance that accepts two instances of the Location structure. The second module is a managed command-line application that imports the GetDistance function, but defines it in terms of a managed equivalent of the Location structure, MLocation. In practice the same name would probably be used for both versions of the structure; however, a different name is used here to demonstrate that the [DllImport](#) prototype is defined in terms of the managed version.

Note that no portion of the DLL is exposed to the managed code using the traditional #include directive. In fact, the DLL is accessed at run time only, so problems with functions imported with DllImport will not be detected at compile time.

### Example: Unmanaged DLL module

```
// TraditionalDll3.cpp
// compile with: /LD /EHsc
#include <iostream>
#include <stdio.h>
#include <math.h>

#define TRADITIONALDLL_EXPORTS
#ifndef TRADITIONALDLL_EXPORTS
    #define TRADITIONALDLL_API __declspec(dllexport)
#else
    #define TRADITIONALDLL_API __declspec(dllimport)
#endif

#pragma pack(push, 8)
struct Location {
    int x;
    int y;
};
#pragma pack(pop)

extern "C" {
    TRADITIONALDLL_API double GetDistance(Location, Location);
    TRADITIONALDLL_API void InitLocation(Location*);
}

double GetDistance(Location loc1, Location loc2) {
    printf_s("[unmanaged] loc1(%d,%d)", loc1.x, loc1.y);
    printf_s(" loc2(%d,%d)\n", loc2.x, loc2.y);

    double h = loc1.x - loc2.x;
    double v = loc1.y - loc2.y;
    double dist = sqrt( pow(h,2) + pow(v,2) );

    return dist;
}

void InitLocation(Location* lp) {
    printf_s("[unmanaged] Initializing location...\n");
    lp->x = 50;
    lp->y = 50;
}
```

### Example: Managed command-line application module

```

// MarshalStruct_pi.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[StructLayout(LayoutKind::Sequential, Pack=8)]
value struct MLocation {
    int x;
    int y;
};

value struct TraditionalDLL {
    [DllImport("TraditionalDLL3.dll")]
    static public double GetDistance(MLocation, MLocation);
    [DllImport("TraditionalDLL3.dll")]
    static public double InitLocation(MLocation*);
};

int main() {
    MLocation loc1;
    loc1.x = 0;
    loc1.y = 0;

    MLocation loc2;
    loc2.x = 100;
    loc2.y = 100;

    double dist = TraditionalDLL::GetDistance(loc1, loc2);
    Console::WriteLine("[managed] distance = {0}", dist);

    MLocation loc3;
    TraditionalDLL::InitLocation(&loc3);
    Console::WriteLine("[managed] x={0} y={1}", loc3.x, loc3.y);
}

```

```

[unmanaged] loc1(0,0) loc2(100,100)
[managed] distance = 141.42135623731
[unmanaged] Initializing location...
[managed] x=50 y=50

```

## See also

[Using Explicit PInvoke in C++ \(DllImport Attribute\)](#)

# How to: Marshal arrays using P/Invoke

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can call native functions that accept C-style strings by using the CLR string type [String](#) when using .NET Framework Platform Invoke (P/Invoke) support. We encourage you to use the C++ Interop features instead of P/Invoke when possible. P/Invoke provides little compile-time error reporting, isn't type-safe, and can be tedious to implement. If the unmanaged API is packaged as a DLL and the source code isn't available, P/Invoke is the only option. Otherwise, see [Using C++ Interop \(Implicit P/Invoke\)](#).

## Example

Because native and managed arrays are laid out differently in memory, passing them successfully across the managed/unmanaged boundary requires conversion, or *marshaling*. This article demonstrates how an array of simple (blitable) items can be passed to native functions from managed code.

As is true of managed/unmanaged data marshaling in general, the [DllImportAttribute](#) attribute is used to create a managed entry point for each native function that's used. In functions that take arrays as arguments, the [MarshalAsAttribute](#) attribute must be used to specify how to marshal the data. In the following example, the [UnmanagedType](#) enumeration is used to indicate that the managed array is marshaled as a C-style array.

The following code consists of an unmanaged and a managed module. The unmanaged module is a DLL that defines a function that accepts an array of integers. The second module is a managed command-line application that imports this function, but defines it in terms of a managed array. It uses the [MarshalAsAttribute](#) attribute to specify that the array should be converted to a native array when called.

```
// TraditionalDll4.cpp
// compile with: /LD /EHsc
#include <iostream>

#define TRADITIONALDLL_EXPORTS
#ifndef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

extern "C" {
    TRADITIONALDLL_API void TakesAnArray(int len, int[]);
}

void TakesAnArray(int len, int a[]) {
    printf_s("[unmanaged]\n");
    for (int i=0; i<len; i++)
        printf("%d = %d\n", i, a[i]);
}
```

The managed module is compiled by using `/clr`.

```
// MarshalBlitArray.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

value struct TraditionalDLL {
    [DllImport("TraditionalDLL4.dll")]
    static public void TakesAnArray(
        int len,[MarshalAs(UnmanagedType::LPArray)]array<int>^);
};

int main() {
    array<int>^ b = gcnew array<int>(3);
    b[0] = 11;
    b[1] = 33;
    b[2] = 55;
    TraditionalDLL::TakesAnArray(3, b);

    Console::WriteLine("[managed]");
    for (int i=0; i<3; i++)
        Console::WriteLine("{0} = {1}", i, b[i]);
}
```

No portion of the DLL is exposed to the managed code through the traditional `#include` directive. In fact, because the DLL is accessed at runtime only, problems in functions imported by using [DllImportAttribute](#) can't be detected at compile time.

## See also

[Using explicit P/Invoke in C++ \(`\[DllImport` attribute\]\)](#)

# How to: Marshal function pointers using P/Invoke

9/20/2022 • 2 minutes to read • [Edit Online](#)

Managed delegates can be used in place of function pointers when interoperating with unmanaged functions by using .NET Framework P/Invoke features. However, we encourage you to use the C++ Interop features instead, when possible. P/Invoke provides little compile-time error reporting, isn't type-safe, and can be tedious to implement. If the unmanaged API is packaged as a DLL and the source code isn't available, P/Invoke is the only option. Otherwise, see these articles:

- [Using C++ Interop \(Implicit P/Invoke\)](#)
- [How to: Marshal callbacks and delegates by using C++ Interop](#)

Unmanaged APIs that take functions pointers as arguments can be called from managed code by using a managed delegate in place of the native function pointer. The compiler automatically marshals the delegate to unmanaged functions as a function pointer. It inserts the necessary managed/unmanaged transition code.

## Example

The following code consists of an unmanaged and a managed module. The unmanaged module is a DLL that defines a function called `TakesCallback` that accepts a function pointer. This address is used to execute the function.

```
// TraditionalDll5.cpp
// compile with: /LD /EHsc
#include <iostream>
#define TRADITIONALDLL_EXPORTS
#ifndef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

extern "C" {
    /* Declare an unmanaged function type that takes two int arguments
       Note the use of __stdcall for compatibility with managed code */
    typedef int (__stdcall *CALLBACK)(int);
    TRADITIONALDLL_API int TakesCallback(CALLBACK fp, int);
}

int TakesCallback(CALLBACK fp, int n) {
    printf_s("[unmanaged] got callback address, calling it...\n");
    return fp(n);
}
```

The managed module defines a delegate that's marshaled to the native code as a function pointer. It uses the `DllImportAttribute` attribute to expose the native `TakesCallback` function to the managed code. In the `main` function, an instance of the delegate is created and passed to the `TakesCallback` function. The program output demonstrates that this function gets executed by the native `TakesCallback` function.

The managed function suppresses garbage collection for the managed delegate to prevent .NET Framework garbage collection from relocating the delegate while the native function executes.

```
// MarshalDelegate.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

public delegate int GetTheAnswerDelegate(int);
public value struct TraditionalDLL {
    [DllImport("TraditionalDLL5.dll")]
    static public int TakesCallback(GetTheAnswerDelegate^ pfn, int n);
};

int GetNumber(int n) {
    Console::WriteLine("[managed] callback!");
    static int x = 0;
    ++x;
    return x + n;
}

int main() {
    GetTheAnswerDelegate^ fp = gcnew GetTheAnswerDelegate(GetNumber);
    pin_ptr<GetTheAnswerDelegate^> pp = &fp;
    Console::WriteLine("[managed] sending delegate as callback...");

    int answer = TraditionalDLL::TakesCallback(fp, 42);
}
```

No portion of the DLL is exposed to the managed code using the traditional `#include` directive. In fact, the DLL is accessed at runtime only, so problems with functions imported by using [DllImportAttribute](#) can't be detected at compile time.

## See also

[Using explicit P/Invoke in C++ \(`DllImport` attribute\)](#)

# How to: Marshal embedded pointers using P/Invoke

9/20/2022 • 2 minutes to read • [Edit Online](#)

Functions that are implemented in unmanaged DLLs can be called from managed code using Platform Invoke (P/Invoke) functionality. If the source code for the DLL isn't available, P/Invoke is the only option for interoperating. However, unlike other .NET languages, Visual C++ provides an alternative to P/Invoke. For more information, see [Using C++ Interop \(Implicit P/Invoke\)](#) and [How to: Marshal embedded pointers using C++ Interop](#).

## Example

Passing structures to native code requires that a managed structure that is equivalent in terms of data layout to the native structure is created. However, structures that contain pointers require special handling. For each embedded pointer in the native structure, the managed version of the structure should contain an instance of the [IntPtr](#) type. Also, memory for these instances must be explicitly allocated, initialized, and released using the [AllocCoTaskMem](#), [StructureToPtr](#), and [FreeCoTaskMem](#) methods.

The following code consists of an unmanaged and a managed module. The unmanaged module is a DLL that defines a function that accepts a structure called `ListStruct` that contains a pointer, and a function called `TakesListStruct`.

```
// TraditionalDll6.cpp
// compile with: /EHsc /LD
#include <stdio.h>
#include <iostream>
#define TRADITIONALDLL_EXPORTS
#ifndef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

#pragma pack(push, 8)
struct ListStruct {
    int count;
    double* item;
};
#pragma pack(pop)

extern "C" {
    TRADITIONALDLL_API void TakesListStruct(ListStruct);
}

void TakesListStruct(ListStruct list) {
    printf_s("[unmanaged] count = %d\n", list.count);
    for (int i=0; i<list.count; i++)
        printf_s("array[%d] = %f\n", i, list.item[i]);
}
```

The managed module is a command-line application that imports the `TakesListStruct` function and defines a structure called `MListStruct` that is equivalent to the native `ListStruct` except that the `double*` is represented with an `IntPtr` instance. Before it calls `TakesListStruct`, the `main` function allocates and initializes the memory that this field references.

```

// EmbeddedPointerMarshalling.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[StructLayout(LayoutKind::Sequential, Pack=8)]
value struct MListStruct {
    int count;
    IntPtr item;
};

value struct TraditionalDLL {
    [DllImport("TraditionalDLL6.dll")]
    static public void TakesListStruct(MListStruct);
};

int main() {
    array<double>^ parray = gcnew array<double>(10);
    Console::WriteLine("[managed] count = {0}", parray->Length);

    Random^ r = gcnew Random();
    for (int i=0; i<parray->Length; i++) {
        parray[i] = r->NextDouble() * 100.0;
        Console::WriteLine("array[{0}] = {1}", i, parray[i]);
    }

    int size = Marshal::SizeOf(double::typeid);
    MListStruct list;
    list.count = parray->Length;
    list.item = Marshal::AllocCoTaskMem(size * parray->Length);

    for (int i=0; i<parray->Length; i++) {
        IntPtr t = IntPtr(list.item.ToInt32() + i * size);
        Marshal::StructureToPtr(parray[i], t, false);
    }

    TraditionalDLL::TakesListStruct( list );
    Marshal::FreeCoTaskMem(list.item);
}

```

No portion of the DLL is exposed to the managed code using the traditional `#include` directive. In fact, the DLL is accessed at runtime only, so problems in functions imported by using [DllImportAttribute](#) can't be detected at compile time.

## See also

[Using explicit P/Invoke in C++ \(`\[DllImport` attribute\]\)](#)

# Using C++ Interop (Implicit PInvoke)

9/20/2022 • 3 minutes to read • [Edit Online](#)

Unlike other .NET languages, Visual C++ has interoperability support that allows managed and unmanaged code to exist in the same application and even in the same file (with the [managed](#), [unmanaged](#) pragmas). This allows Visual C++ developers to integrate .NET functionality into existing Visual C++ applications without disturbing the rest of the application.

You can also call unmanaged functions from a managed compiland using [\\_\\_declspec\(dllexport\)](#), [\\_\\_declspec\(dllimport\)](#).

Implicit PInvoke is useful when you do not need to specify how function parameters will be marshaled, or any of the other details that can be specified when explicitly calling [DllImportAttribute](#).

Visual C++ provides two ways for managed and unmanaged functions to interoperate:

- [Using Explicit PInvoke in C++ \(DllImport Attribute\)](#)

Explicit PInvoke is supported by the .NET Framework and is available in most .NET languages. But as its name implies, C++ Interop is specific to Visual C++.

## C++ Interop

C++ Interop provides better type safety, and it is typically less tedious to implement. However, C++ Interop is not an option if the unmanaged source code is not available, or for cross-platform projects.

## C++ COM Interop

The interoperability features supported by Visual C++ offer a particular advantage over other .NET languages when it comes to interoperating with COM components. Instead of being limited to the restrictions of the .NET Framework [Tlbimp.exe \(Type Library Importer\)](#), such as limited support for data types and the mandatory exposure of every member of every COM interface, C++ Interop allows COM components to be accessed at will and does not require separate interop assemblies. Unlike Visual Basic and C#, Visual C++ can use COM objects directly using the usual COM mechanisms (such as [CoCreateInstance](#) and [QueryInterface](#)). This is possible due to C++ Interop features that cause the compiler to automatically insert the transition code to move from managed to unmanaged functions and back again.

Using C++ Interop, COM components can be used as they are normally used or they can be wrapped inside C++ classes. These wrapper classes are called custom runtime callable wrappers, or CRCWs, and they have two advantages over using COM directly in application code:

- The resulting class can be used from languages other than Visual C++.
- The details of the COM interface can be hidden from the managed client code. .NET data types can be used in place of native types, and the details of data marshaling can be performed transparently inside the CRCW.

Regardless of whether COM is used directly or through a CRCW, argument types other than simple, blittable types must be marshaled.

## Blittable Types

For unmanaged APIs that use simple, intrinsic types (see [Blittable and Non-Blittable Types](#)), no special coding is required because these data types have the same representation in memory, but more complex data types

require explicit data marshaling. For an example, see [How to: Call Native DLLs from Managed Code Using PInvoke](#).

## Example

```
// vcmcppv2_impl_dllimp.cpp
// compile with: /clr:pure user32.lib
using namespace System::Runtime::InteropServices;

// Implicit DLLImport specifying calling convention
extern "C" int __stdcall MessageBeep(int);

// explicit DLLImport needed here to use P/Invoke marshalling because
// System::String ^ is not the type of the first parameter to printf
[DllImport("msvcrt.dll", EntryPoint = "printf", CallingConvention = CallingConvention::Cdecl, CharSet =
CharSet::Ansi)]
// or just
// [DllImport("msvcrt.dll")]
int printf(System::String ^, ...);

int main() {
    // (string literals are System::String by default)
    printf("Begin beep\n");
    MessageBeep(100000);
    printf("Done\n");
}
```

```
Begin beep
Done
```

## In This Section

- [How to: Marshal ANSI Strings Using C++ Interop](#)
- [How to: Marshal Unicode Strings Using C++ Interop](#)
- [How to: Marshal COM Strings Using C++ Interop](#)
- [How to: Marshal Structures Using C++ Interop](#)
- [How to: Marshal Arrays Using C++ Interop](#)
- [How to: Marshal Callbacks and Delegates By Using C++ Interop](#)
- [How to: Marshal Embedded Pointers Using C++ Interop](#)
- [How to: Access Characters in a System::String](#)
- [How to: Convert char \\* String to System::Byte Array](#)
- [How to: Convert System::String to wchar\\_t\\* or char\\*](#)
- [How to: Convert System::String to Standard String](#)
- [How to: Convert Standard String to System::String](#)
- [How to: Obtain a Pointer to Byte Array](#)
- [How to: Load Unmanaged Resources into a Byte Array](#)
- [How to: Modify Reference Class in a Native Function](#)

- [How to: Determine if an Image is Native or CLR](#)
- [How to: Add Native DLL to Global Assembly Cache](#)
- [How to: Hold Reference to Value Type in Native Type](#)
- [How to: Hold Object Reference in Unmanaged Memory](#)
- [How to: Detect /clr Compilation](#)
- [How to: Convert Between System::Guid and \\_GUID](#)
- [How to: Specify an out Parameter](#)
- [How to: Use a Native Type in a /clr Compilation](#)
- [How to: Declare Handles in Native Types](#)
- [How to: Wrap Native Class for Use by C#](#)

For information on using delegates in an interop scenario, see [delegate \(C++ Component Extensions\)](#).

## See also

- [Calling Native Functions from Managed Code](#)

# How to: Marshal ANSI Strings Using C++ Interop

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic demonstrates how ANSI strings can be passed using C++ Interop, but the .NET Framework [String](#) represents strings in Unicode format, so conversion to ANSI is an extra step. For interoperating with other string types, see the following topics:

- [How to: Marshal Unicode Strings Using C++ Interop](#)
- [How to: Marshal COM Strings Using C++ Interop](#)

The following code examples use the [managed](#), [unmanaged](#) #pragma directives to implement managed and unmanaged functions in the same file, but these functions interoperate in the same manner if defined in separate files. Because files containing only unmanaged functions do not need to be compiled with [/clr](#) ([Common Language Runtime Compilation](#)), they can retain their performance characteristics.

## Example: Pass ANSI string

The example demonstrates passing an ANSI string from a managed to an unmanaged function using [StringToHGlobalAnsi](#). This method allocates memory on the unmanaged heap and returns the address after performing the conversion. This means that no pinning is necessary (because memory on the GC heap is not being passed to the unmanaged function) and that the IntPtr returned from [StringToHGlobalAnsi](#) must be explicitly released or a memory leak results.

```
// MarshalANSI1.cpp
// compile with: /clr
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

void NativeTakesAString(const char* p) {
    printf_s("(native) received '%s'\n", p);
}

#pragma managed

int main() {
    String^ s = gcnew String("sample string");
    IntPtr ip = Marshal::StringToHGlobalAnsi(s);
    const char* str = static_cast<const char*>(ip.ToPointer());

    Console::WriteLine("(managed) passing string...");
    NativeTakesAString( str );

    Marshal::FreeHGlobal( ip );
}
```

## Example: Data marshaling required to access ANSI string

The following example demonstrates the data marshaling required to access an ANSI string in a managed

function that is called by an unmanaged function. The managed function, on receiving the native string, can either use it directly or convert it to a managed string using the [PtrToStringAnsi](#) method, as shown.

```
// MarshalANSI2.cpp
// compile with: /clr
#include <iostream>
#include <vcclr.h>

using namespace std;

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma managed

void ManagedStringFunc(char* s) {
    String^ ms = Marshal::PtrToStringAnsi(static_cast<IntPtr>(s));
    Console::WriteLine("(managed): received '{0}'", ms);
}

#pragma unmanaged

void NativeProvidesAString() {
    cout << "(native) calling managed func...\n";
    ManagedStringFunc("test string");
}

#pragma managed

int main() {
    NativeProvidesAString();
}
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Marshal Unicode Strings Using C++ Interop

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic demonstrates one facet of Visual C++ interoperability. For more information, see [Using C++ Interop \(Implicit PInvoke\)](#).

The following code examples use the `managed`, `unmanaged` #pragma directives to implement managed and unmanaged functions in the same file, but these functions interoperate in the same manner if defined in separate files. Files containing only unmanaged functions do not need to be compiled with [/clr \(Common Language Runtime Compilation\)](#).

This topic demonstrates how Unicode strings can be passed from a managed to an unmanaged function, and vice versa. For interoperating with other strings types, see the following topics:

- [How to: Marshal ANSI Strings Using C++ Interop](#)
- [How to: Marshal COM Strings Using C++ Interop](#)

## Example: Pass Unicode string from managed to unmanaged function

To pass a Unicode string from a managed to an unmanaged function, the `PtrToStringChars` function (declared in `Vcclr.h`) can be used to access in the memory where the managed string is stored. Because this address will be passed to a native function, it is important that the memory be pinned with [pin\\_ptr \(C++/CLI\)](#) to prevent the string data from being relocated, should a garbage collection cycle take place while the unmanaged function executes.

```
// MarshalUnicode1.cpp
// compile with: /clr
#include <iostream>
#include <stdio.h>
#include <vcclr.h>

using namespace std;

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

void NativeTakesAString(const wchar_t* p) {
    printf_s("(native) received '%S'\n", p);
}

#pragma managed

int main() {
    String^ s = gcnew String("test string");
    pin_ptr<const wchar_t> str = PtrToStringChars(s);

    Console::WriteLine("(managed) passing string to native func...");
    NativeTakesAString( str );
}
```

## Example: Data marshaling required to access Unicode string

The following example demonstrates the data marshaling required to access a Unicode string in a managed function called by an unmanaged function. The managed function, on receiving the native Unicode string, converts it to a managed string using the [PtrToStringUni](#) method.

```
// MarshalUnicode2.cpp
// compile with: /clr
#include <iostream>

using namespace std;
using namespace System;
using namespace System::Runtime::InteropServices;

#pragma managed

void ManagedStringFunc(wchar_t* s) {
    String^ ms = Marshal::PtrToStringUni((IntPtr)s);
    Console::WriteLine("(managed) received '{0}'", ms);
}

#pragma unmanaged

void NativeProvidesAString() {
    cout << "(unmanaged) calling managed func...\n";
    ManagedStringFunc(L"test string");
}

#pragma managed

int main() {
    NativeProvidesAString();
}
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Marshal COM Strings Using C++ Interop

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic demonstrates how a BSTR (the basic string format favored in COM programming) can be passed from a managed to an unmanaged function, and vice versa. For interoperating with other strings types, see the following topics:

- [How to: Marshal Unicode Strings Using C++ Interop](#)
- [How to: Marshal ANSI Strings Using C++ Interop](#)

The following code examples use the `managed`, `unmanaged` #pragma directives to implement managed and unmanaged functions in the same file, but these functions interoperate in the same manner if defined in separate files. Files containing only unmanaged functions do not need to be compiled with [/clr \(Common Language Runtime Compilation\)](#).

## Example: Pass BSTR from managed to unmanaged function

The following example demonstrates how a BSTR (a string format used in COM programming) can be passed from a managed to an unmanaged function. The calling managed function uses `StringToBSTR` to obtain the address of a BSTR representation of the contents of a .NET `System.String`. This pointer is pinned using `pin_ptr` ([C++/CLI](#)) to ensure that its physical address is not changed during a garbage collection cycle while the unmanaged function executes. The garbage collector is prohibited from moving the memory until the `pin_ptr` ([C++/CLI](#)) goes out of scope.

```
// MarshalBSTR1.cpp
// compile with: /clr
#define WINVER 0x0502
#define _AFXDLL
#include <afxwin.h>

#include <iostream>
using namespace std;

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

void NativeTakesAString(BSTR bstr) {
    printf_s("%S", bstr);
}

#pragma managed

int main() {
    String^ s = "test string";

    IntPtr ip = Marshal::StringToBSTR(s);
    BSTR bs = static_cast<BSTR>(ip.ToPointer());
    pin_ptr<BSTR> b = &bs;

    NativeTakesAString( bs );
    Marshal::FreeBSTR(ip);
}
```

## Example: Pass BSTR from unmanaged to managed function

The following example demonstrates how a BSTR can be passed from an unmanaged to a managed function. The receiving managed function can either use the string in as a BSTR or use [PtrToStringBSTR](#) to convert it to a [String](#) for use with other managed functions. Because the memory representing the BSTR is allocated on the unmanaged heap, no pinning is necessary, because there is no garbage collection on the unmanaged heap.

```
// MarshalBSTR2.cpp
// compile with: /clr
#define WINVER 0x0502
#define _AFXDLL
#include <afxwin.h>

#include <iostream>
using namespace std;

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma managed

void ManagedTakesAString(BSTR bstr) {
    String^ s = Marshal::PtrToStringBSTR(static_cast<IntPtr>(bstr));
    Console::WriteLine("(managed) converted BSTR to String: '{0}'", s);
}

#pragma unmanaged

void UnManagedFunc() {
    BSTR bs = SysAllocString(L"test string");
    printf_s("(unmanaged) passing BSTR to managed func...\n");
    ManagedTakesAString(bs);
}

#pragma managed

int main() {
    UnManagedFunc();
}
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Marshal Structures Using C++ Interop

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic demonstrates one facet of Visual C++ interoperability. For more information, see [Using C++ Interop \(Implicit PInvoke\)](#).

The following code examples use the `managed`, `unmanaged` #pragma directives to implement managed and unmanaged functions in the same file, but these functions interoperate in the same manner if defined in separate files. Files containing only unmanaged functions do not need to be compiled with [/clr \(Common Language Runtime Compilation\)](#).

## Example: Pass structure from managed to unmanaged function

The following example demonstrates passing a structure from a managed to an unmanaged function, both by value and by reference. Because the structure in this example contains only simple, intrinsic data types (see [Blittable and Non-Blittable Types](#)), no special marshaling is required. To marshal non-blittable structures, such as those that contain pointers, see [How to: Marshal Embedded Pointers Using C++ Interop](#).

```

// PassStruct1.cpp
// compile with: /clr

#include <stdio.h>
#include <math.h>

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

struct Location {
    int x;
    int y;
};

double GetDistance(Location loc1, Location loc2) {
    printf_s("[unmanaged] loc1(%d,%d)", loc1.x, loc1.y);
    printf_s(" loc2(%d,%d)\n", loc2.x, loc2.y);

    double h = loc1.x - loc2.x;
    double v = loc1.y - loc2.y;
    double dist = sqrt( pow(h,2) + pow(v,2) );

    return dist;
}

void InitLocation(Location* lp) {
    printf_s("[unmanaged] Initializing location...\n");
    lp->x = 50;
    lp->y = 50;
}

#pragma managed

int main() {
    Location loc1;
    loc1.x = 0;
    loc1.y = 0;

    Location loc2;
    loc2.x = 100;
    loc2.y = 100;

    double dist = GetDistance(loc1, loc2);
    Console::WriteLine("[managed] distance = {0}", dist);

    Location loc3;
    InitLocation(&loc3);
    Console::WriteLine("[managed] x={0} y={1}", loc3.x, loc3.y);
}

```

## Example: Pass structure from unmanaged to managed function

The following example demonstrates passing a structure from an unmanaged to a managed function, both by value and by reference. Because the structure in this example contains only simple, intrinsic data types (see [Blittable and Non-Blittable Types](#)), no special marshalling is required. To marshal non-blittable structures, such as those that contain pointers, see [How to: Marshal Embedded Pointers Using C++ Interop](#).

```

// PassStruct2.cpp
// compile with: /clr
#include <stdio.h>
#include <math.h>
using namespace System;

// native structure definition
struct Location {
    int x;
    int y;
};

#pragma managed

double GetDistance(Location loc1, Location loc2) {
    Console::Write("[managed] got loc1({0},{1})", loc1.x, loc1.y);
    Console::WriteLine(" loc2({0},{1})", loc2.x, loc2.y);

    double h = loc1.x - loc2.x;
    double v = loc1.y = loc2.y;
    double dist = sqrt( pow(h,2) + pow(v,2) );

    return dist;
}

void InitLocation(Location* lp) {
    Console::WriteLine("[managed] Initializing location...");
    lp->x = 50;
    lp->y = 50;
}

#pragma unmanaged

int UnmanagedFunc() {
    Location loc1;
    loc1.x = 0;
    loc1.y = 0;

    Location loc2;
    loc2.x = 100;
    loc2.y = 100;

    printf_s("(unmanaged) loc1=(%d,%d)", loc1.x, loc1.y);
    printf_s(" loc2=(%d,%d)\n", loc2.x, loc2.y);

    double dist = GetDistance(loc1, loc2);
    printf_s("[unmanaged] distance = %f\n", dist);

    Location loc3;
    InitLocation(&loc3);
    printf_s("[unmanaged] got x=%d y=%d\n", loc3.x, loc3.y);

    return 0;
}

#pragma managed

int main() {
    UnmanagedFunc();
}

```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Marshal Arrays Using C++ Interop

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic demonstrates one facet of Visual C++ interoperability. For more information, see [Using C++ Interop \(Implicit PInvoke\)](#).

The following code examples use the [managed](#), [unmanaged](#) #pragma directives to implement managed and unmanaged functions in the same file, but these functions interoperate in the same manner if defined in separate files. Files containing only unmanaged functions do not need to be compiled with [/clr \(Common Language Runtime Compilation\)](#).

## Example: Pass managed array to unmanaged function

The following example demonstrates how to pass a managed array to an unmanaged function. The managed function uses [pin\\_ptr \(C++/CLI\)](#) to suppress garbage collection for the array before calling the unmanaged function. By providing the unmanaged function with a pinned pointer into the GC heap, the overhead of making a copy of the array can be avoided. To demonstrate that the unmanaged function is accessing GC heap memory, it modifies the contents of the array and the changes are reflected when the managed function resumes control.

```

// PassArray1.cpp
// compile with: /clr
#ifndef _CRT_RAND_S
#define _CRT_RAND_S
#endif

#include <iostream>
#include <stdlib.h>
using namespace std;

using namespace System;

#pragma unmanaged

void TakesAnArray(int* a, int c) {
    cout << "(unmanaged) array received:\n";
    for (int i=0; i<c; i++)
        cout << "a[" << i << "] = " << a[i] << "\n";

    unsigned int number;
    errno_t err;

    cout << "(unmanaged) modifying array contents...\n";
    for (int i=0; i<c; i++) {
        err = rand_s( &number );
        if ( err == 0 )
            a[i] = number % 100;
    }
}

#pragma managed

int main() {
    array<int>^ nums = gcnew array<int>(5);

    nums[0] = 0;
    nums[1] = 1;
    nums[2] = 2;
    nums[3] = 3;
    nums[4] = 4;

    Console::WriteLine("(managed) array created:");
    for (int i=0; i<5; i++)
        Console::WriteLine("a[{0}] = {1}", i, nums[i]);

    pin_ptr<int> pp = &nums[0];
    TakesAnArray(pp, 5);

    Console::WriteLine("(managed) contents:");
    for (int i=0; i<5; i++)
        Console::WriteLine("a[{0}] = {1}", i, nums[i]);
}

```

## Example: Pass unmanaged array to managed function

The following example demonstrates passing an unmanaged array to a managed function. The managed function accesses the array memory directly (as opposed to creating a managed array and copying the array content), which allows changes made by the managed function to be reflected in the unmanaged function when it regains control.

```

// PassArray2.cpp
// compile with: /clr
#include <iostream>
using namespace std;

using namespace System;

#pragma managed

void ManagedTakesAnArray(int* a, int c) {
    Console::WriteLine("(managed) array received:");
    for (int i=0; i<c; i++)
        Console::WriteLine("a[{0}] = {1}", i, a[i]);

    cout << "(managed) modifying array contents...\n";
    Random^ r = gcnew Random(DateTime::Now.Second);
    for (int i=0; i<c; i++)
        a[i] = r->Next(100);
}

#pragma unmanaged

void NativeFunc() {
    int nums[5] = { 0, 1, 2, 3, 4 };

    printf_s("(unmanaged) array created:\n");
    for (int i=0; i<5; i++)
        printf_s("a[%d] = %d\n", i, nums[i]);

    ManagedTakesAnArray(nums, 5);

    printf_s("(unmanaged) contents:\n");
    for (int i=0; i<5; i++)
        printf_s("a[%d] = %d\n", i, nums[i]);
}

#pragma managed

int main() {
    NativeFunc();
}

```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Marshal Callbacks and Delegates By Using C++ Interop

9/20/2022 • 3 minutes to read • [Edit Online](#)

This topic demonstrates the marshalling of callbacks and delegates (the managed version of a callback) between managed and unmanaged code using Visual C++.

The following code examples use the [managed](#), [unmanaged](#) #pragma directives to implement managed and unmanaged functions in the same file, but the functions could also be defined in separate files. Files containing only unmanaged functions do not need to be compiled with the [/clr \(Common Language Runtime Compilation\)](#).

## Example: Configure unmanaged API to trigger managed delegate

The following example demonstrates how to configure an unmanaged API to trigger a managed delegate. A managed delegate is created and one of the interop methods, [GetFunctionPointerForDelegate](#), is used to retrieve the underlying entry point for the delegate. This address is then passed to the unmanaged function, which calls it with no knowledge of the fact that it is implemented as a managed function.

Notice that it is possible, but not necessary, to pin the delegate using [pin\\_ptr \(C++/CLI\)](#) to prevent it from being re-located or disposed of by the garbage collector. Protection from premature garbage collection is needed, but pinning provides more protection than is necessary, as it prevents collection but also prevents relocation.

If a delegate is re-located by a garbage collection, it will not affect the underlying managed callback, so [Alloc](#) is used to add a reference to the delegate, allowing relocation of the delegate, but preventing disposal. Using [GCHandle](#) instead of [pin\\_ptr](#) reduces fragmentation potential of the managed heap.

```

// MarshalDelegate1.cpp
// compile with: /clr
#include <iostream>

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

// Declare an unmanaged function type that takes two int arguments
// Note the use of __stdcall for compatibility with managed code
typedef int (__stdcall *ANSWERCB)(int, int);

int TakesCallback(ANSWERCB fp, int n, int m) {
    printf_s("[unmanaged] got callback address, calling it...\n");
    return fp(n, m);
}

#pragma managed

public delegate int GetTheAnswerDelegate(int, int);

int GetNumber(int n, int m) {
    Console::WriteLine("[managed] callback!");
    return n + m;
}

int main() {
    GetTheAnswerDelegate^ fp = gcnew GetTheAnswerDelegate(GetNumber);
    GCHandle gch = GCHandle::Alloc(fp);
    IntPtr ip = Marshal::GetFunctionPointerForDelegate(fp);
    ANSWERCB cb = static_cast<ANSWERCB>(ip.ToPointer());
    Console::WriteLine("[managed] sending delegate as callback...");

    // force garbage collection cycle to prove
    // that the delegate doesn't get disposed
    GC::Collect();

    int answer = TakesCallback(cb, 243, 257);

    // release reference to delegate
    gch.Free();
}

```

## Example: Function pointer stored by unmanaged API

The following example is similar to the previous example, but in this case the provided function pointer is stored by the unmanaged API, so it can be invoked at any time, requiring that garbage collection be suppressed for an arbitrary length of time. As a result, the following example uses a global instance of [GCHandle](#) to prevent the delegate from being relocated, independent of function scope. As discussed in the first example, using `pin_ptr` is unnecessary for these examples, but in this case wouldn't work anyway, as the scope of a `pin_ptr` is limited to a single function.

```

// MarshalDelegate2.cpp
// compile with: /clr
#include <iostream>

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

// Declare an unmanaged function type that takes two int arguments
// Note the use of __stdcall for compatibility with managed code
typedef int (__stdcall *ANSWERCB)(int, int);
static ANSWERCB cb;

int TakesCallback(ANSWERCB fp, int n, int m) {
    cb = fp;
    if (cb) {
        printf_s("[unmanaged] got callback address (%d), calling it...\n", cb);
        return cb(n, m);
    }
    printf_s("[unmanaged] unregistering callback");
    return 0;
}

#pragma managed

public delegate int GetTheAnswerDelegate(int, int);

int GetNumber(int n, int m) {
    Console::WriteLine("[managed] callback!");
    static int x = 0;
    ++x;

    return n + m + x;
}

static GCHandle gch;

int main() {
    GetTheAnswerDelegate^ fp = gcnew GetTheAnswerDelegate(GetNumber);

    gch = GCHandle::Alloc(fp);

    IntPtr ip = Marshal::GetFunctionPointerForDelegate(fp);
    ANSWERCB cb = static_cast<ANSWERCB>(ip.ToPointer());
    Console::WriteLine("[managed] sending delegate as callback...");

    int answer = TakesCallback(cb, 243, 257);

    // possibly much later (in another function)...

    Console::WriteLine("[managed] releasing callback mechanisms...");
    TakesCallback(0, 243, 257);
    gch.Free();
}

```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Marshal Embedded Pointers Using C++ Interop

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following code examples use the `managed`, `unmanaged` #pragma directives to implement managed and unmanaged functions in the same file, but these functions interoperate in the same manner if defined in separate files. Files containing only unmanaged functions do not need to be compiled with [/clr \(Common Language Runtime Compilation\)](#).

## Example

The following example demonstrates how an unmanaged function that takes a structure containing pointers can be called from a managed function. The managed function creates an instance of the structure and initializes the embedded pointer with the `new` keyword (instead of the `ref new`, `gcnew` keyword). Because this allocates the memory on the native heap, there is no need to pin the array to suppress garbage collection. However, the memory must be explicitly deleted to avoid memory leakage.

```
// marshal_embedded_pointer.cpp
// compile with: /clr
#include <iostream>

using namespace System;
using namespace System::Runtime::InteropServices;

// unmanaged struct
struct ListStruct {
    int count;
    double* item;
};

#pragma unmanaged

void UnmanagedTakesListStruct(ListStruct list) {
    printf_s("[unmanaged] count = %d\n", list.count);
    for (int i=0; i<list.count; i++)
        printf_s("array[%d] = %f\n", i, list.item[i]);
}

#pragma managed

int main() {
    ListStruct list;
    list.count = 10;
    list.item = new double[list.count];

    Console::WriteLine("[managed] count = {0}", list.count);
    Random^ r = gcnew Random(0);
    for (int i=0; i<list.count; i++) {
        list.item[i] = r->NextDouble() * 100.0;
        Console::WriteLine("array[{0}] = {1}", i, list.item[i]);
    }

    UnmanagedTakesListStruct( list );
    delete list.item;
}
```

```
[managed] count = 10
array[0] = 72.624326996796
array[1] = 81.7325359590969
array[2] = 76.8022689394663
array[3] = 55.8161191436537
array[4] = 20.6033154021033
array[5] = 55.8884794618415
array[6] = 90.6027066011926
array[7] = 44.2177873310716
array[8] = 97.754975314138
array[9] = 27.370445768987
[unmanaged] count = 10
array[0] = 72.624327
array[1] = 81.732536
array[2] = 76.802269
array[3] = 55.816119
array[4] = 20.603315
array[5] = 55.888479
array[6] = 90.602707
array[7] = 44.217787
array[8] = 97.754975
array[9] = 27.370446
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Extend the Marshaling Library

9/20/2022 • 5 minutes to read • [Edit Online](#)

This topic explains how to extend the marshaling library to provide more conversions between data types. Users can extend the marshaling library for any data conversions not currently supported by the library.

You can extend the marshaling library in one of two ways - with or without a [marshal\\_context Class](#). Review the [Overview of Marshaling in C++](#) topic to determine whether a new conversion requires a context.

In both cases, you first create a file for new marshaling conversions. You do so to preserve the integrity of the standard marshaling library files. If you want to port a project to another computer or to another programmer, you must copy the new marshaling file together with the rest of the project. In this manner, the user receiving the project will be guaranteed to receive the new conversions and will not have to modify any library files.

## To Extend the Marshaling Library with a Conversion that does not Require a Context

1. Create a file to store the new marshaling functions, for example, MyMarshal.h.
2. Include one or more of the marshal library files:
  - marshal.h for base types.
  - marshal\_windows.h for windows data types.
  - marshal\_cppstd.h for C++ Standard Library data types.
  - marshal\_atl.h for ATL data types.
3. Use the code at the end of these steps to write the conversion function. In this code, TO is the type to convert to, FROM is the type to convert from, and `from` is the parameter to be converted.
4. Replace the comment about conversion logic with code to convert the `from` parameter into an object of TO type and return the converted object.

```
namespace msclr {
    namespace interop {
        template<>
        inline TO marshal_as<TO, FROM> (const FROM& from) {
            // Insert conversion logic here, and return a TO parameter.
        }
    }
}
```

## To Extend the Marshaling Library with a Conversion that Requires a Context

1. Create a file to store the new marshaling functions, for example, MyMarshal.h
2. Include one or more of the marshal library files:
  - marshal.h for base types.
  - marshal\_windows.h for windows data types.
  - marshal\_cppstd.h for C++ Standard Library data types.
  - marshal\_atl.h for ATL data types.
3. Use the code at the end of these steps to write the conversion function. In this code, TO is the type to

convert to, FROM is the type to convert from, `toObject` is a pointer in which to store the result, and `fromObject` is the parameter to be converted.

4. Replace the comment about initializing with code to initialize the `toPtr` to the appropriate empty value. For example, if it is a pointer, set it to `NULL`.
5. Replace the comment about conversion logic with code to convert the `from` parameter into an object of `TO` type. This converted object will be stored in `toPtr`.
6. Replace the comment about setting `toObject` with code to set `toObject` to your converted object.
7. Replace the comment about cleaning up native resources with code to free any memory allocated by `toPtr`. If `toPtr` allocated memory by using `new`, use `delete` to free the memory.

```
namespace msclr {
    namespace interop {
        template<>
        ref class context_node<TO, FROM> : public context_node_base
        {
        private:
            TO toPtr;
        public:
            context_node(TO& toObject, FROM fromObject)
            {
                // (Step 4) Initialize toPtr to the appropriate empty value.
                // (Step 5) Insert conversion logic here.
                // (Step 6) Set toObject to the converted parameter.
            }
            ~context_node()
            {
                this->!context_node();
            }
        protected:
            !context_node()
            {
                // (Step 7) Clean up native resources.
            }
        };
    }
}
```

## Example: Extend marshaling library

The following example extends the marshaling library with a conversion that does not require a context. In this example, the code converts the employee information from a native data type to a managed data type.

```

// MyMarshalNoContext.cpp
// compile with: /clr
#include <msclr/marshal.h>

value struct ManagedEmp {
    System::String^ name;
    System::String^ address;
    int zipCode;
};

struct NativeEmp {
    char* name;
    char* address;
    int zipCode;
};

namespace msclr {
    namespace interop {
        template<>
        inline ManagedEmp^ marshal_as<ManagedEmp^, NativeEmp> (const NativeEmp& from) {
            ManagedEmp^ toValue = gcnew ManagedEmp;
            toValue->name = marshal_as<System::String^>(from.name);
            toValue->address = marshal_as<System::String^>(from.address);
            toValue->zipCode = from.zipCode;
            return toValue;
        }
    }
}

using namespace System;
using namespace msclr::interop;

int main() {
    NativeEmp employee;

    employee.name = "Jeff Smith";
    employee.address = "123 Main Street";
    employee.zipCode = 98111;

    ManagedEmp^ result = marshal_as<ManagedEmp^>(employee);

    Console::WriteLine("Managed name: {0}", result->name);
    Console::WriteLine("Managed address: {0}", result->address);
    Console::WriteLine("Managed zip code: {0}", result->zipCode);

    return 0;
}

```

In the previous example, the `marshal_as` function returns a handle to the converted data. This was done in order to prevent creating an additional copy of the data. Returning the variable directly would have an unnecessary performance cost associated with it.

```

Managed name: Jeff Smith
Managed address: 123 Main Street
Managed zip code: 98111

```

## Example: Convert employee information

The following example converts the employee information from a managed data type to a native data type. This conversion requires a marshaling context.

```
// MyMarshalContext.cpp
```

```

// compile with: /clr
#include <stdlib.h>
#include <string.h>
#include <msclr/marshal.h>

value struct ManagedEmp {
    System::String^ name;
    System::String^ address;
    int zipCode;
};

struct NativeEmp {
    const char* name;
    const char* address;
    int zipCode;
};

namespace msclr {
    namespace interop {
        template<>
        ref class context_node<NativeEmp*, ManagedEmp^> : public context_node_base
        {
        private:
            NativeEmp* toPtr;
            marshal_context context;
        public:
            context_node(NativeEmp*& toObject, ManagedEmp^ fromObject)
            {
                // Conversion logic starts here
                toPtr = NULL;

                const char* nativeName;
                const char* nativeAddress;

                // Convert the name from String^ to const char*.
                System::String^ tempValue = fromObject->name;
                nativeName = context.marshal_as<const char*>(tempValue);

                // Convert the address from String^ to const char*.
                tempValue = fromObject->address;
                nativeAddress = context.marshal_as<const char*>(tempValue);

                toPtr = new NativeEmp();
                toPtr->name = nativeName;
                toPtr->address = nativeAddress;
                toPtr->zipCode = fromObject->zipCode;

                toObject = toPtr;
            }
            ~context_node()
            {
                this->!context_node();
            }
        protected:
            !context_node()
            {
                // When the context is deleted, it will free the memory
                // allocated for toPtr->name and toPtr->address, so toPtr
                // is the only memory that needs to be freed.
                if (toPtr != NULL) {
                    delete toPtr;
                    toPtr = NULL;
                }
            }
        };
    };
}

using namespace System;

```

```
using namespace msclr::interop;

int main() {
    ManagedEmp^ employee = gcnew ManagedEmp();

    employee->name = gcnew String("Jeff Smith");
    employee->address = gcnew String("123 Main Street");
    employee->zipCode = 98111;

    marshal_context context;
    NativeEmp* result = context.marshal_as<NativeEmp*>(employee);

    if (result != NULL) {
        printf_s("Native name: %s\nNative address: %s\nNative zip code: %d\n",
            result->name, result->address, result->zipCode);
    }

    return 0;
}
```

```
Native name: Jeff Smith
Native address: 123 Main Street
Native zip code: 98111
```

## See also

[Overview of Marshaling in C++](#)

# How to: Access Characters in a System::String

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can access characters of a `String` object for high-performance calls to unmanaged functions that take `wchar_t*` strings. The method yields an interior pointer to the first character of the `String` object. This pointer can be manipulated directly or pinned and passed to a function expecting an ordinary `wchar_t` string.

## Examples

`PtrToStringChars` returns a `Char`, which is an interior pointer (also known as a `byref`). As such, it is subject to garbage collection. You don't have to pin this pointer unless you're going to pass it to a native function.

Consider the following code. Pinning is not needed because `ppchar` is an interior pointer, and if the garbage collector moves the string it points to, it will also update `ppchar`. Without a `pin_ptr (C++/CLI)`, the code will work and not have the potential performance hit caused by pinning.

If you pass `ppchar` to a native function, then it must be a pinning pointer; the garbage collector will not be able to update any pointers on the unmanaged stack frame.

```
// PtrToStringChars.cpp
// compile with: /clr
#include<vcclr.h>
using namespace System;

int main() {
    String ^ mystring = "abcdefg";

    interior_ptr<const Char> ppchar = PtrToStringChars( mystring );

    for ( ; *ppchar != L'\0'; ++ppchar )
        Console::Write(*ppchar);
}
```

```
abcdefg
```

This example shows where pinning is needed.

```

// PtrToStringChars_2.cpp
// compile with: /clr
#include <string.h>
#include <vcclr.h>
// using namespace System;

size_t getlen(System::String ^ s) {
    // Since this is an outside string, we want to be secure.
    // To be secure, we need a maximum size.
    size_t maxsize = 256;
    // make sure it doesn't move during the unmanaged call
    pin_ptr<const wchar_t> pinchars = PtrToStringChars(s);
    return wcsnlen(pinchars, maxsize);
};

int main() {
    System::Console::WriteLine(getlen("testing"));
}

```

7

An interior pointer has all the properties of a native C++ pointer. For example, you can use it to walk a linked data structure and do insertions and deletions using only one pointer:

```

// PtrToStringChars_3.cpp
// compile with: /clr /LD
using namespace System;
ref struct ListNode {
    Int32 elem;
    ListNode ^ Next;
};

void deleteNode( ListNode ^ list, Int32 e ) {
    interior_ptr<ListNode ^> ptrToNext = &list;
    while ( *ptrToNext != nullptr ) {
        if ( (*ptrToNext) -> elem == e )
            *ptrToNext = (*ptrToNext) -> Next;    // delete node
        else
            ptrToNext = &(*ptrToNext) -> Next;    // move to next node
    }
}

```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Convert char \* String to System::Byte Array

9/20/2022 • 2 minutes to read • [Edit Online](#)

The most efficient way to convert a `char *` string to a `Byte` array is to use `Marshal` class.

## Example

```
// convert_native_string_to_Byte_array.cpp
// compile with: /clr
#include <string.h>

using namespace System;
using namespace System::Runtime::InteropServices;

int main() {
    char buf[] = "Native String";
    int len = strlen(buf);

    array< Byte >^ byteArray = gcnew array< Byte >(len + 2);

    // convert native pointer to System::IntPtr with C-Style cast
    Marshal::Copy((IntPtr)buf,byteArray, 0, len);

    for ( int i = byteArray->GetLowerBound(0); i <= byteArray->GetUpperBound(0); i++ ) {
        char dc = *(Byte^) byteArray->GetValue(i);
        Console::Write((Char)dc);
    }

    Console::WriteLine();
}
```

```
Native String
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Convert System::String to wchar\_t\* or char\*

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can use `PtrToStringChars` in `Vcclr.h` to convert `String` to native `wchar_t *` or `char *`. This always returns a wide Unicode string pointer because CLR strings are internally Unicode. You can then convert from wide as shown in the following example.

## Example

```
// convert_string_to_wchar.cpp
// compile with: /clr
#include < stdio.h >
#include < stdlib.h >
#include < vcclr.h >

using namespace System;

int main() {
    String ^str = "Hello";

    // Pin memory so GC can't move it while native function is called
    pin_ptr<const wchar_t> wch = PtrToStringChars(str);
    printf_s("%S\n", wch);

    // Conversion to char* :
    // Can just convert wchar_t* to char* using one of the
    // conversion functions such as:
    // WideCharToMultiByte()
    // wcstombs_s()
    // ... etc
    size_t convertedChars = 0;
    size_t sizeInBytes = ((str->Length + 1) * 2);
    errno_t err = 0;
    char *ch = (char *)malloc(sizeInBytes);

    err = wcstombs_s(&convertedChars,
                     ch, sizeInBytes,
                     wch, sizeInBytes);
    if (err != 0)
        printf_s("wcstombs_s failed!\n");

    printf_s("%s\n", ch);
}
```

```
Hello
Hello
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Convert System::String to Standard String

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can convert a `String` to `std::string` or `std::wstring`, without using `PtrToStringChars` in Vclr.h.

## Example

```
// convert_system_string.cpp
// compile with: /clr
#include <string>
#include <iostream>
using namespace std;
using namespace System;

void MarshalString ( String ^ s, string& os ) {
    using namespace Runtime::InteropServices;
    const char* chars =
        (const char*)(Marshal::StringToHGlobalAnsi(s)).ToPointer();
    os = chars;
    Marshal::FreeHGlobal(IntPtr((void*)chars));
}

void MarshalString ( String ^ s, wstring& os ) {
    using namespace Runtime::InteropServices;
    const wchar_t* chars =
        (const wchar_t*)(Marshal::StringToHGlobalUni(s)).ToPointer();
    os = chars;
    Marshal::FreeHGlobal(IntPtr((void*)chars));
}

int main() {
    string a = "test";
    wstring b = L"test2";
    String ^ c = gcnew String("abcd");

    cout << a << endl;
    MarshalString(c, a);
    c = "efgh";
    MarshalString(c, b);
    cout << a << endl;
    wcout << b << endl;
}
```

```
test
abcd
efgh
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Convert Standard String to System::String

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic shows how convert a C++ Standard Library string (`<string>`) to a [String](#).

## Example

```
// convert_standard_string_to_system_string.cpp
// compile with: /clr
#include <string>
#include <iostream>
using namespace System;
using namespace std;

int main() {
    string str = "test";
    cout << str << endl;
    String^ str2 = gcnew String(str.c_str());
    Console::WriteLine(str2);

    // alternatively
    String^ str3 = gcnew String(str.data());
    Console::WriteLine(str3);
}
```

```
test
test
test
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Obtain a Pointer to Byte Array

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can obtain a pointer to the array block in a `Byte` array by taking the address of the first element and assigning it to a pointer.

## Example

```
// pointer_to_Byte_array.cpp
// compile with: /clr
using namespace System;
int main() {
    Byte bArr[] = {1, 2, 3};
    Byte* pbArr = &bArr[0];

    array<Byte> ^ bArr2 = gcnew array<Byte>{1,2,3};
    interior_ptr<Byte> pbArr2 = &bArr2[0];
}
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Load Unmanaged Resources into a Byte Array

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic discusses several ways to load unmanaged resources into a [Byte](#) array.

## Examples

If you know the size of your unmanaged resource, you can preallocate a CLR array and then load the resource into the array using a pointer to the array block of the CLR array.

```
// load_unmanaged_resources_into_Byte_array.cpp
// compile with: /clr
using namespace System;
void unmanaged_func( unsigned char * p ) {
    for ( int i = 0; i < 10; i++ )
        p[ i ] = i;
}

public ref class A {
public:
    void func() {
        array<Byte> ^b = gcnew array<Byte>(10);
        pin_ptr<Byte> p =  &b[ 0 ];
        Byte * np = p;
        unmanaged_func( np );    // pass pointer to the block of CLR array.
        for ( int i = 0; i < 10; i++ )
            Console::Write( b[ i ] );
        Console::WriteLine();
    }
};

int main() {
    A^ g = gcnew A;
    g->func();
}
```

0123456789

This sample shows how to copy data from an unmanaged memory block to a managed array.

```
// load_unmanaged_resources_into_Byte_array_2.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

#include <string.h>
int main() {
    char buf[] = "Native String";
    int len = strlen(buf);
    array<Byte> ^byteArray = gcnew array<Byte>(len + 2);

    // convert any native pointer to IntPtr by doing C-Style cast
    Marshal::Copy( (IntPtr)buf, byteArray, 0, len );
}
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Modify Reference Class in a Native Function

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can pass a reference class with a CLR array to a native function, and modify the class, using PInvoke services.

## Examples

Compile the following native library.

```
// modify_ref_class_in_native_function.cpp
// compile with: /LD
#include <stdio.h>
#include <windows.h>

struct S {
    wchar_t* str;
    int intarr[2];
};

extern "C" {
    __declspec(dllexport) int bar(S* param) {
        printf_s("str: %S\n", param->str);
        fflush(stdin);
        fflush(stdout);
        printf_s("In native: intarr: %d, %d\n",
            param->intarr[0], param->intarr[1]);
        fflush(stdin);
        fflush(stdout);
        param->intarr[0]=300;param->intarr[1]=400;
        return 0;
    }
}
```

Compile the following assembly.

```
// modify_ref_class_in_native_function_2.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[StructLayout(LayoutKind::Sequential, CharSet = CharSet::Unicode)]
ref class S {
public:
    [MarshalAs(UnmanagedType::LPWStr)]
    String ^ str;
    [MarshalAs(UnmanagedType::ByValArray,
        ArraySubType=UnmanagedType::I4, SizeConst=2)]
    array<Int32> ^ intarr;
};

[DllImport("modify_ref_class_in_native_function.dll",
    CharSet=CharSet::Unicode)]
int bar([In][Out] S ^ param);

int main() {
    S ^ param = gcnew S;
    param->str = "Hello";
    param->intarr = gcnew array<Int32>(2);
    param->intarr[0] = 100;
    param->intarr[1] = 200;
    bar(param); // Call to native function
    Console::WriteLine("In managed: intarr: {0}, {1}",
        param->intarr[0], param->intarr[1]);
}
```

```
str: Hello
In native: intarr: 100, 200
In managed: intarr: 300, 400
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Determine if an Image is Native or CLR

9/20/2022 • 2 minutes to read • [Edit Online](#)

One way to determine whether an image was built for the common language runtime is to use [dumpbin/CLRHEADER](#).

You can also programmatically check whether an image was built for the common language runtime. For more information, see [How to: Detect /clr Compilation](#).

## Example

The following sample determines whether an image was built to run on the common language runtime.

```

// detect_image_type.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;

enum class CompilationMode {Invalid, Native, CLR };

static CompilationMode IsManaged(String^ filename) {
    try {
        array<Byte>^ data = gcnew array<Byte>(4096);
        FileInfo^ file = gcnew FileInfo(filename);
        Stream^ fin = file->Open(FileMode::Open, FileAccess::Read);
        Int32 iRead = fin->Read(data, 0, 4096);
        fin->Close();

        // Verify this is a executable/dll
        if ((data[1] << 8 | data[0]) != 0x5a4d)
            return CompilationMode::Invalid;

        // This will get the address for the WinNT header
        Int32 iWinNTHdr = data[63]<<24 | data[62]<<16 | data[61] << 8 | data[60];

        // Verify this is an NT address
        if ((data[iWinNTHdr+3] << 24 | data[iWinNTHdr+2] << 16 | data[iWinNTHdr+1] << 8 | data[iWinNTHdr]) != 0x00004550)
            return CompilationMode::Invalid;

        Int32 iLightningAddr = iWinNTHdr + 24 + 208;
        Int32 iSum = 0;
        Int32 iTop = iLightningAddr + 8;

        for (int i = iLightningAddr; i < iTop; ++i)
            iSum |= data[i];

        if (iSum == 0)
            return CompilationMode::Native;
        else
            return CompilationMode::CLR;
    }
    catch(Exception ^e) {
        throw(e);
    }
}

int main() {
    array<String^>^ args = Environment::GetCommandLineArgs();

    if (args->Length < 2) {
        Console::WriteLine("USAGE : detect_clr <assembly_name>\n");
        return -1;
    }

    Console::WriteLine("{0} is compiled {1}", args[1], IsManaged(args[1]));
}

```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Add Native DLL to Global Assembly Cache

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can put a native DLL (not COM) into the Global Assembly Cache.

## Example

/ASSEMBLYLINKRESOURCE lets you embed a native DLL in an assembly.

For more information, see [/ASSEMBLYLINKRESOURCE \(Link to .NET Framework Resource\)](#).

```
/ASSEMBLYLINKRESOURCE:MyComponent.dll
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Hold Reference to Value Type in Native Type

9/20/2022 • 2 minutes to read • [Edit Online](#)

Use `gcroot` on the boxed type to hold a reference to a value type in a native type.

## Example

```
// reference_to_value_in_native.cpp
// compile with: /clr
#using <mscorlib.dll>
#include <vcclr.h>

using namespace System;

public value struct V {
    String ^str;
};

class Native {
public:
    gcroot< V^ > v_handle;
};

int main() {
    Native native;
    V v;
    native.v_handle = v;
    native.v_handle->str = "Hello";
    Console::WriteLine("String in V: {0}", native.v_handle->str);
}
```

```
String in V: Hello
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Hold Object Reference in Unmanaged Memory

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can use `gcroot.h`, which wraps `GCHandle`, to hold a CLR object reference in unmanaged memory. Alternatively, you can use `GCHandle` directly.

## Examples

```
// hold_object_reference.cpp
// compile with: /clr
#include "gcroot.h"
using namespace System;

#pragma managed
class StringWrapper {

private:
    gcroot<String ^> x;

public:
    StringWrapper() {
        String ^ str = gcnew String("ManagedString");
        x = str;
    }

    void PrintString() {
        String ^ targetStr = x;
        Console::WriteLine("StringWrapper::x == {0}", targetStr);
    }
};

#pragma unmanaged
int main() {
    StringWrapper s;
    s.PrintString();
}
```

```
StringWrapper::x == ManagedString
```

`GCHandle` gives you a means to hold a managed object reference in unmanaged memory. You use the `Alloc` method to create an opaque handle to a managed object and `Free` to release it. Also, the `Target` method allows you to obtain the object reference back from the handle in managed code.

```
// hold_object_reference_2.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

#pragma managed
class StringWrapper {
    IntPtr m_handle;
public:
    StringWrapper() {
        String ^ str = gcnew String("ManagedString");
        m_handle = static_cast<IntPtr>(GCHandle::Alloc(str));
    }
    ~StringWrapper() {
        static_cast<GCHandle>(m_handle).Free();
    }

    void PrintString() {
        String ^ targetStr = safe_cast< String ^ >(static_cast<GCHandle>(m_handle).Target);
        Console::WriteLine("StringWrapper::m_handle == {0}", targetStr);
    }
};

#pragma unmanaged
int main() {
    StringWrapper s;
    s.PrintString();
}
```

```
StringWrapper::m_handle == ManagedString
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Detect /clr Compilation

9/20/2022 • 2 minutes to read • [Edit Online](#)

Use the `_MANAGED` or `_M_CEE` macro to see if a module is compiled with `/clr`. For more information, see [/clr \(Common Language Runtime Compilation\)](#).

For more information about macros, see [Predefined Macros](#).

## Example

```
// detect_CLR_compilation.cpp
// compile with: /clr
#include <stdio.h>

int main() {
    #if (_MANAGED == 1) || (_M_CEE == 1)
        printf_s("compiling with /clr\n");
    #else
        printf_s("compiling without /clr\n");
    #endif
}
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Convert Between System::Guid and \_GUID

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following code sample shows how to convert between a [Guid](#) and a [\\_GUID](#).

## Example

```
// convert_guids.cpp
// compile with: /clr
#include <windows.h>
#include <stdio.h>

using namespace System;

Guid FromGUID( _GUID& guid ) {
    return Guid( guid.Data1, guid.Data2, guid.Data3,
                 guid.Data4[ 0 ], guid.Data4[ 1 ],
                 guid.Data4[ 2 ], guid.Data4[ 3 ],
                 guid.Data4[ 4 ], guid.Data4[ 5 ],
                 guid.Data4[ 6 ], guid.Data4[ 7 ] );
}

_GUID ToGUID( Guid& guid ) {
    array<Byte>^ guidData = guid.ToByteArray();
    pin_ptr<Byte> data = &(guidData[ 0 ]);

    return *(_GUID *)data;
}

int main() {
    _GUID ng = {0x11111111,0x2222,0x3333,0x44,0x55,0x55,0x55,0x55,0x55,0x55};
    Guid mg;

    Console::WriteLine( (mg = FromGUID( ng )).ToString() );
    _GUID ng2 = ToGUID( mg );

    printf_s( "%x-%x-%x-", ng2.Data1, ng2.Data2, ng2.Data3 );
    for (int i = 0 ; i < 8 ; i++) {
        if (i == 2)
            printf_s("-");
        printf_s("%x", ng2.Data4[i]);
    }
    printf_s("\n");
}
```

```
11111111-2222-3333-4455-555555555555
11111111-2222-3333-4455-555555555555
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Specify an out parameter

9/20/2022 • 2 minutes to read • [Edit Online](#)

This sample shows how to specify that a function parameter is an `out` parameter, and how to call that function from a C# program.

An `out` parameter is specified in C++ by using [OutAttribute](#).

## Example

The first part of this sample creates a C++ DLL. It defines a type that contains a function with an `out` parameter.

```
// cpp_out_param.cpp
// compile with: /LD /clr
using namespace System;
public value struct TestStruct {
    static void Test([Runtime::InteropServices::Out] String^ %s) {
        s = "a string";
    }
};
```

This source file is a C# client that consumes the C++ component created in the previous example.

```
// cpp_out_param_2.cs
// compile with: /reference:cpp_out_param.dll
using System;
class TestClass {
    public static void Main() {
        String t;
        TestStruct.Test(out t);
        System.Console.WriteLine(t);
    }
}
```

```
a string
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Use a Native Type in a /clr Compilation

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can define a native type in a /clr compilation and any use of that native type from within the assembly is valid. However, native types will not be available for use from referenced metadata.

Each assembly must contain the definition of every native type it will use.

For more information, see [/clr \(Common Language Runtime Compilation\)](#).

## Examples

This sample creates a component that defines and uses a native type.

```
// use_native_type_in_clr.cpp
// compile with: /clr /LD
public struct NativeClass {
    static int Test() { return 98; }
};

public ref struct ManagedClass {
    static int i = NativeClass::Test();
    void Test() {
        System::Console::WriteLine(i);
    }
};
```

This sample defines a client that consumes the component. Notice that it is an error to access the native type, unless it is defined in the compiland.

```
// use_native_type_in_clr_2.cpp
// compile with: /clr
#using "use_native_type_in_clr.dll"
// Uncomment the following 3 lines to resolve.
// public struct NativeClass {
//     static int Test() { return 98; }
// };

int main() {
    ManagedClass x;
    x.Test();

    System::Console::WriteLine(NativeClass::Test()); // C2653
}
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Declare Handles in Native Types

9/20/2022 • 2 minutes to read • [Edit Online](#)

You cannot declare a handle type in a native type. vcclr.h provides the type-safe wrapper template `gcroot` to refer to a CLR object from the C++ heap. This template lets you embed a virtual handle in a native type and treat it as if it were the underlying type. In most cases, you can use the `gcroot` object as the embedded type without any casting. However, with `for each, in`, you have to use `static_cast` to retrieve the underlying managed reference.

The `gcroot` template is implemented using the facilities of the value class `System::Runtime::InteropServices::GCHandle`, which provides "handles" into the garbage-collected heap. Note that the handles themselves are not garbage collected and are freed when no longer in use by the destructor in the `gcroot` class (this destructor cannot be called manually). If you instantiate a `gcroot` object on the native heap, you must call `delete` on that resource.

The runtime will maintain an association between the handle and the CLR object, which it references. When the CLR object moves with the garbage-collected heap, the handle will return the new address of the object. A variable does not have to be pinned before it is assigned to a `gcroot` template.

## Examples

This sample shows how to create a `gcroot` object on the native stack.

```
// mcpp_gcroot.cpp
// compile with: /clr
#include <vcclr.h>
using namespace System;

class CppClass {
public:
    gcroot<String^> str; // can use str as if it were String^
    CppClass() {}
};

int main() {
    CppClass c;
    c.str = gcnew String("hello");
    Console::WriteLine( c.str ); // no cast required
}
```

```
hello
```

This sample shows how to create a `gcroot` object on the native heap.

```

// mcpp_gcroot_2.cpp
// compile with: /clr
// compile with: /clr
#include <vcclr.h>
using namespace System;

struct CppClass {
    gcroot<String ^> * str;
    CppClass() : str(new gcroot<String ^>) {}

    ~CppClass() { delete str; }

};

int main() {
    CppClass c;
    *c.str = gcnew String("hello");
    Console::WriteLine( *c.str );
}

```

hello

This sample shows how to use `gcroot` to hold references to value types (not reference types) in a native type by using `gcroot` on the boxed type.

```

// mcpp_gcroot_3.cpp
// compile with: /clr
#include < vcclr.h >
using namespace System;

public value struct V {
    String^ str;
};

class Native {
public:
    gcroot< V^ > v_handle;
};

int main() {
    Native native;
    V v;
    native.v_handle = v;
    native.v_handle->str = "Hello";
    Console::WriteLine("String in V: {0}", native.v_handle->str);
}

```

String in V: Hello

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# How to: Wrap Native Class for Use by C#

9/20/2022 • 2 minutes to read • [Edit Online](#)

This sample shows how to wrap a native C++ class so it can be consumed by code authored in C#, or other .NET language.

## Example

```
// wrap_native_class_for_mgd_consumption.cpp
// compile with: /clr /LD
#include <windows.h>
#include <vcclr.h>
#using <System.dll>

using namespace System;

class UnmanagedClass {
public:
    LPCWSTR GetPropertyA() { return 0; }
    void MethodB( LPCWSTR ) {}
};

public ref class ManagedClass {
public:
    // Allocate the native object on the C++ Heap via a constructor
    ManagedClass() : m_Impl( new UnmanagedClass() {} )

    // Deallocate the native object on a destructor
    ~ManagedClass() {
        delete m_Impl;
    }

protected:
    // Deallocate the native object on the finalizer just in case no destructor is called
    !ManagedClass() {
        delete m_Impl;
    }

public:
    property String ^ get_PropertyA {
        String ^ get() {
            return gcnew String( m_Impl->GetPropertyA() );
        }
    }

    void MethodB( String ^ theString ) {
        pin_ptr<const WCHAR> str = PtrToStringChars(theString);
        m_Impl->MethodB(str);
    }

private:
    UnmanagedClass * m_Impl;
};

}
```

## See also

[Using C++ Interop \(Implicit PInvoke\)](#)

# Pure and verifiable code (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

For .NET Programming, Visual C++ in Visual Studio 2017 supports the creation of mixed assemblies by using the [/clr \(Common Language Runtime Compilation\)](#) compiler option. The `/clr:pure` and `clr:safe` options are deprecated in Visual Studio 2015 and unsupported in Visual Studio 2017. If your code needs to be safe or verifiable, then we recommend that you port it to C#.

## Mixed (/clr)

Mixed assemblies (compiled with `/clr`), contain both unmanaged and managed parts, making it possible for them to use .NET features, but still contain native code. This allows applications and components to be updated to use .NET features without requiring that the entire project be rewritten. Using Visual C++ to mix managed and native code in this fashion is called C++ Interop. For more information, see [Mixed \(Native and Managed Assemblies\)](#) and [Native and .NET Interoperability](#).

Calls made from managed assemblies to native DLLs via P/Invoke will compile, but may fail at runtime depending on security settings.

There is one coding scenario that will pass the compiler but that will result in an unverifiable assembly: calling a virtual function through an object instance using the scope resolution operator. For example:

```
MyObj -> A::VirtualFunction(); .
```

## See also

- [.NET Programming with C++/CLI \(Visual C++\)](#)

# How to: create verifiable C++ projects (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Visual C++ application wizards do not create verifiable projects.

## IMPORTANT

Visual Studio 2015 deprecated and Visual Studio 2017 does not support the `/clr:pure` and `/clr:safe` creation of verifiable projects. If you require verifiable code, we recommend you translate your code to C#.

However, if you are using an older version of the Microsoft C++ compiler toolset that supports `/clr:pure` and `/clr:safe`, projects can be converted to be verifiable. This topic describes how to set project properties and modify project source files to transform your Visual Studio C++ projects to produce verifiable applications.

## Compiler and linker settings

By default, .NET projects use the `/clr` compiler flag and configure the linker to target x86 hardware. For verifiable code, you must use the `/clr:safe` flag, and you must instruct the linker to generate MSIL instead of native machine instructions.

### To change the compiler and linker settings

1. Display the project Property Page. For more information, see [Set compiler and build properties](#).
2. On the **General** page under the **Configuration Properties** node, set the **Common Language Runtime Support** property to **Safe MSIL Common Language Runtime Support (/clr:safe)**.
3. On the **Advanced** page under the **Linker** node, set the **CLR Image Type** property to **Force safe IL image (/CLRIMAGETYPE:SAFE)**.

## Removing native data types

Because native data types are non-verifiable, even if they are not actually used, you must remove all header files containing native types.

## NOTE

The following procedure applies to Windows Forms Application (.NET) and Console Application (.NET) projects.

### To remove references to native data types

1. Comment out everything in the `Stdafx.h` file.

## Configuring an entry point

Because verifiable applications cannot use the C run-time libraries (CRT), they cannot depend on the CRT to call the main function as the standard entry point. This means that you must explicitly provide the name of the function to be called initially to the linker. (In this case, `Main()` is used instead of `main()` or `_tmain()` to indicate a non-CRT entry point, but because the entry point must be specified explicitly, this name is arbitrary.)

**NOTE**

The following procedures apply to Console Application (.NET) projects.

**To configure an entry point**

1. Change `_tmain()` to `Main()` in the project's main .cpp file.
2. Display the project Property Page. For more information, see [Set compiler and build properties](#).
3. On the **Advanced** page under the **Linker** node, enter `Main` as the **Entry Point** property value.

## See also

- [Pure and Verifiable Code \(C++/CLI\)](#)

# Using Verifiable Assemblies with SQL Server (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Extended stored procedures, packaged as dynamic-link libraries (DLLs), provide a way to extend SQL Server functionality through functions developed with Visual C++. Extended stored procedures are implemented as functions inside DLLs. In addition to functions, extended stored procedures can also define [user-defined types](#) and aggregate functions (such as SUM or AVG).

When a client executes an extended stored procedure, SQL Server searches for the DLL associated with the extended stored procedure and loads the DLL. SQL Server calls the requested extended stored procedure and executes it under a specified security context. The extended stored procedure then passes result sets and returns parameters back to the server.

SQL Server provides extensions to Transact-SQL (T-SQL) to allow you to install verifiable assemblies into SQL Server. The SQL Server permission set specifies the security context, with the following levels of security:

- Unrestricted mode: Run code at your own risk; code does not have to be verifiably type-safe.
- Safe mode: Run verifiably typesafe code; compiled with /clr:safe.

## IMPORTANT

Visual Studio 2015 deprecated and Visual Studio 2017 does not support the `/clr:pure` and `/clr:safe` creation of verifiable projects. If you require verifiable code, we recommend you translate your code to C#.

To create and load a verifiable assembly into SQL Server, use the Transact-SQL commands CREATE ASSEMBLY and DROP ASSEMBLY as follows:

```
CREATE ASSEMBLY <assemblyName> FROM <'Assembly UNC Path'> WITH  
    PERMISSION_SET <permissions>  
DROP ASSEMBLY <assemblyName>
```

The PERMISSION\_SET command specifies the security context, and can have the values UNRESTRICTED, SAFE, or EXTENDED.

In addition, you can use the CREATE FUNCTION command to bind to method names in a class:

```
CREATE FUNCTION <FunctionName>(<FunctionParams>)  
RETURNS returnType  
[EXTERNAL NAME <AssemblyName>:<ClassName>:<StaticMethodName>]
```

## Example

The following SQL script (for example, named "MyScript.sql") loads an assembly into SQL Server and makes a method of a class available:

```
-- Create assembly without external access
drop assembly stockNoEA
go
create assembly stockNoEA
from
'c:\stockNoEA.dll'
with permission_set safe

-- Create function on assembly with no external access
drop function GetQuoteNoEA
go
create function GetQuoteNoEA(@sym nvarchar(10))
returns real
external name stockNoEA:StockQuotes::GetQuote
go

-- To call the function
select dbo.GetQuoteNoEA('MSFT')
go
```

SQL scripts can be executed interactively in SQL Query Analyzer or at the command line with the sqlcmd.exe utility. The following command line connects to MyServer, uses the default database, uses a trusted connection, inputs MyScript.sql, and outputs MyResult.txt.

```
sqlcmd -S MyServer -E -i myScript.sql -o myResult.txt
```

## See also

[Classes and Structs](#)

# Converting projects from mixed mode to pure intermediate language

9/20/2022 • 2 minutes to read • [Edit Online](#)

All Visual C++ CLR projects link to the C run-time libraries by default. As a result, these projects are classified as *mixed-mode* applications, because they combine native code with code that targets the common language runtime (managed code). When they're compiled, they get compiled into intermediate language (IL), also known as Microsoft intermediate language (MSIL).

## IMPORTANT

Visual Studio 2015 deprecated and Visual Studio 2017 no longer supports the creation of `/clr:pure` or `/clr:safe` code for CLR applications. If you require pure or safe assemblies, we recommend you translate your application to C#.

If you're using an earlier version of the Microsoft C++ compiler toolset that supports `/clr:pure` or `/clr:safe`, you can use this procedure to convert your code to pure MSIL:

## To convert your mixed-mode application into pure intermediate language

1. Remove links to the [C runtime libraries](#) (CRT):

- a. In the .cpp file defining the entry point of your application, change the entry point to `Main()`. Using `Main()` indicates that your project doesn't link to the CRT.
- b. In Solution Explorer, right-click your project and select **Properties** on the shortcut menu to open the property pages for your application.
- c. In the **Advanced** project property page for the **Linker**, select the **Entry Point** and then enter `Main` in this field.
- d. For console applications, in the **System** project property page for the **Linker**, select the **SubSystem** field and change it to `Console (/SUBSYSTEM:CONSOLE)`.

## NOTE

You don't have to set this property for Windows Forms applications because the **SubSystem** field is set to `Windows (/SUBSYSTEM:WINDOWS)` by default.

- e. In `stdafx.h`, comment out all the `#include` statements. For example, in console applications:

```
// #include <iostream>
// #include <tchar.h>
```

-or-

For example, in Windows Forms applications:

```
// #include <stdlib.h>
// #include <malloc.h>
// #include <memory.h>
// #include <tchar.h>
```

- f. For Windows Forms applications, in `Form1.cpp`, comment out the `#include` statement that references `windows.h`. For example:

```
// #include <windows.h>
```

2. Add the following code to `stdafx.h`:

```
#ifndef __FLTUSED__
#define __FLTUSED__
    extern "C" __declspec(selectany) int _fltused=1;
#endif
```

3. Remove all unmanaged types:

Wherever appropriate, replace unmanaged types with references to structures from the [System](#) namespace. Common managed types are listed in the following table:

STRUCTURE	DESCRIPTION
<code>Boolean</code>	Represents a Boolean value.
<code>Byte</code>	Represents an 8-bit unsigned integer.
<code>Char</code>	Represents a Unicode character.
<code>DateTime</code>	Represents an instant in time, typically expressed as a date and time of day.
<code>Decimal</code>	Represents a decimal number.
<code>Double</code>	Represents a double-precision floating-point number.
<code>Guid</code>	Represents a globally unique identifier (GUID).
<code>Int16</code>	Represents a 16-bit signed integer.
<code>Int32</code>	Represents a 32-bit signed integer.
<code>Int64</code>	Represents a 64-bit signed integer.
<code>IntPtr</code>	A platform-specific type that is used to represent a pointer or a handle.
<code>SByte</code>	Represents an 8-bit signed integer.
<code>Single</code>	Represents a single-precision floating-point number.

STRUCTURE	DESCRIPTION
<code>TimeSpan</code>	Represents a time interval.
<code>UInt16</code>	Represents a 16-bit unsigned integer.
<code>UInt32</code>	Represents a 32-bit unsigned integer.
<code>UInt64</code>	Represents a 64-bit unsigned integer.
<code>UIntPtr</code>	A platform-specific type that is used to represent a pointer or a handle.
<code>Void</code>	Indicates a method that doesn't return a value; that is, the method has the <code>void</code> return type.

# Serialization (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Serialization (the process of storing the state of an object or member to a permanent medium) of managed classes (including individual fields or properties) is supported by the **SerializableAttribute** and **NonSerializedAttribute** classes.

## Remarks

Apply the **SerializableAttribute** custom attribute to a managed class to serialize the entire class or apply only to specific fields or properties to serialize parts of the managed class. Use the **NonSerializedAttribute** custom attribute to exempt fields or properties of a managed class from being serialized.

## Example

### Description

In the following example, the class `MyClass` (and the property `m_nCount`) is marked as serializable. However, the `m_nData` property is not serialized as indicated by the **NonSerialized** custom attribute:

### Code

```
// serialization_and_mcpp.cpp
// compile with: /LD /clr
using namespace System;

[ Serializable ]
public ref class MyClass {
public:
    int m_nCount;
private:
    [ NonSerialized ]
    int m_nData;
};
```

### Comments

Note that both attributes can be referenced using their "short name" (**Serializable** and **NonSerialized**). This is further explained in [Applying Attributes](#).

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

# Friend Assemblies (C++)

9/20/2022 • 3 minutes to read • [Edit Online](#)

For applicable runtimes, the *friend assemblies* language feature makes types that are at namespace scope or global scope in an assembly component accessible to one or more client assemblies or .netmodules.

## All Runtimes

### Remarks

(This language feature is not supported in all runtimes.)

## Windows Runtime

### Remarks

(This language feature is not supported in the Windows Runtime.)

### Requirements

Compiler option: /ZW

## Common Language Runtime

### Remarks

**To make types at namespace scope or global scope in an assembly component accessible to a client assembly or .netmodule**

1. In the component, specify an assembly attribute `InternalsVisibleToAttribute`, and pass the name of the client assembly or .netmodule that will access types at namespace scope or global scope in the component. You can specify multiple client assemblies or .netmodules by specifying additional attributes.
2. In the client assembly or .netmodule, when you reference the component assembly by using `#using`, pass the `as_friend` attribute. If you specify the `as_friend` attribute for an assembly that does not specify `InternalsVisibleToAttribute`, a runtime exception will be thrown if you try to access a type at namespace scope or global scope in the component.

A build error will result if the assembly that contains the `InternalsVisibleToAttribute` attribute does not have a strong name but the client assembly that uses the `as_friend` attribute does.

Although types at namespace scope and global scope can be known to a client assembly or .netmodule, member accessibility is still in effect. For example, you cannot access a private member.

Access to all types in an assembly must be explicitly granted. For example, assembly C does not have access to all types in assembly A if assembly C references assembly B and assembly B has access to all types in assembly A.

For information about how to sign—that is, how to give a strong name to—an assembly that is built by using the Microsoft C++ compiler, see [Strong Name Assemblies \(Assembly Signing\) \(C++/CLI\)](#).

As an alternative to using the friend assemblies feature, you can use `StrongNameIdentityPermission` to restrict access to individual types.

### Requirements

Compiler option: /clr

## Examples

The following code example defines a component that specifies a client assembly that has access to the types in the component.

```
// friend_assemblies.cpp
// compile by using: /clr /LD
using namespace System::Runtime::CompilerServices;
using namespace System;
// an assembly attribute, not bound to a type
[assembly:InternalsVisibleTo("friend_assemblies_2")];

ref class Class1 {
public:
    void Test_Public() {
        Console::WriteLine("Class1::Test_Public");
    }
};
```

The next code example accesses a private type in the component.

```
// friend_assemblies_2.cpp
// compile by using: /clr
#using "friend_assemblies.dll" as_friend

int main() {
    Class1 ^ a = gcnew Class1;
    a->Test_Public();
}
```

```
Class1::Test_Public
```

The next code example defines a component but does not specify a client assembly that will have access to the types in the component.

Notice that the component is linked by using `/opt:noref`. This ensures that private types are emitted in the component's metadata, which is not required when the `InternalsVisibleTo` attribute is present. For more information, see [/OPT \(Optimizations\)](#).

```
// friend_assemblies_3.cpp
// compile by using: /clr /LD /link /opt:noref
using namespace System;

ref class Class1 {
public:
    void Test_Public() {
        Console::WriteLine("Class1::Test_Public");
    }
};
```

The following code example defines a client that tries to access a private type in a component that does not give access to its private types. Because of the behavior of the runtime, if you want to catch the exception, you must attempt to access a private type in a helper function.

```

// friend_assemblies_4.cpp
// compile by using: /clr
#using "friend_assemblies_3.dll" as_friend
using namespace System;

void Test() {
    Class1 ^ a = gcnew Class1;
}

int main() {
    // to catch this kind of exception, use a helper function
    try {
        Test();
    }
    catch(MethodAccessException ^ e) {
        Console::WriteLine("caught an exception");
    }
}

```

caught an exception

The next code example shows how to create a strong-name component that specifies a client assembly that will have access to the types in the component.

```

// friend_assemblies_5.cpp
// compile by using: /clr /LD /link /keyfile:friend_assemblies.snk
using namespace System::Runtime::CompilerServices;
using namespace System;
// an assembly attribute, not bound to a type

[assembly:InternalsVisibleTo("friend_assemblies_6,
PublicKey=0024000048000009400000060200000024000052534131000400001000100bf45d77fd991f3bfff0ef51af48a12d356
99e04616f27ba561195a69ebd3449c345389dc9603d65be8cd1987bc7ea48bdda35ac7d57d3d82c666b7fc1a5b79836d139ef0ac8c4e
715434211660f481612771a9f7059b9b742c3d8af00e01716ed4b872e6f1be0e94863eb5745224f0deaba5b137624d7049b6f2d87fba
639fc5")];

private ref class Class1 {
public:
    void Test_Public() {
        Console::WriteLine("Class1::Test_Public");
    }
};

```

Notice that the component must specify its public key. We suggest that you run the following commands sequentially at a command prompt to create a key pair and get the public key:

**sn -d friend\_assemblies.snk**

**sn -k friend\_assemblies.snk**

**sn -i friend\_assemblies.snk friend\_assemblies.snk**

**sn -pc friend\_assemblies.snk key.publickey**

**sn -tp key.publickey**

The next code example accesses a private type in the strong-name component.

```
// friend_assemblies_6.cpp
// compile by using: /clr /link /keyfile:friend_assemblies.snk
#using "friend_assemblies_5.dll" as_friend

int main() {
    Class1 ^ a = gcnew Class1;
    a->Test_Public();
}
```

```
Class1::Test_Public
```

## See also

[Component Extensions for Runtime Platforms](#)

# Managed Types (C++/CLI)

9/20/2022 • 5 minutes to read • [Edit Online](#)

Visual C++ allows access to .NET features through managed types, which provide support for features of the common language runtime and are subject to the advantages and restrictions of the runtime.

## Managed Types and the main Function

When you write an application using `/clr`, the arguments of the `main()` function can't be of a managed type.

An example of a proper signature is:

```
// managed_types_and_main.cpp
// compile with: /clr
int main(int, char*[], char*[]){}
```

## .NET Framework Equivalents to C++ Native Types

The following table shows the keywords for built-in Visual C++ types, which are aliases of predefined types in the **System** namespace.

VISUAL C++ TYPE	.NET FRAMEWORK TYPE
<code>void</code>	<code>System.Void</code>
<code>bool</code>	<code>System.Boolean</code>
<code>signed char</code>	<code>System.SByte</code>
<code>unsigned char</code>	<code>System.Byte</code>
<code>wchar_t</code>	<code>System.Char</code>
<code>short</code> and <code>signed short</code>	<code>System.Int16</code>
<code>unsigned short</code>	<code>System.UInt16</code>
<code>int</code> , <code>signed int</code> , <code>long</code> , and <code>signed long</code>	<code>System.Int32</code>
<code>unsigned int</code> and <code>unsigned long</code>	<code>System.UInt32</code>
<code>__int64</code> and <code>signed __int64</code>	<code>System.Int64</code>
<code>unsigned __int64</code>	<code>System.UInt64</code>
<code>float</code>	<code>System.Single</code>
<code>double</code> and <code>long double</code>	<code>System.Double</code>

For more information about the compiler option to default to `signed char` or `unsigned char`, see [/J \(Default char type is unsigned\)](#).

## Version Issues for Value Types Nested in Native Types

Consider a signed (strong name) assembly component used to build a client assembly. The component contains a value type that is used in the client as the type for a member of a native union, a class, or an array. If a future version of the component changes the size or layout of the value type, the client must be recompiled.

Create a keyfile with `sn.exe ( sn -k mykey.snk )`.

### Example

The following sample is the component.

```
// nested_value_types.cpp
// compile with: /clr /LD
using namespace System::Reflection;
[assembly:AssemblyVersion("1.0.0.*"),
assembly:AssemblyKeyFile("mykey.snk")];

public value struct S {
    int i;
    void Test() {
        System::Console::WriteLine("S.i = {0}", i);
    }
};
```

This sample is the client:

```
// nested_value_types_2.cpp
// compile with: /clr
#using <nested_value_types.dll>

struct S2 {
    S MyS1, MyS2;
};

int main() {
    S2 MyS2a, MyS2b;
    MyS2a.MyS1.i = 5;
    MyS2a.MyS2.i = 6;
    MyS2b.MyS1.i = 10;
    MyS2b.MyS2.i = 11;

    MyS2a.MyS1.Test();
    MyS2a.MyS2.Test();
    MyS2b.MyS1.Test();
    MyS2b.MyS2.Test();
}
```

The example produces this output:

```
S.i = 5
S.i = 6
S.i = 10
S.i = 11
```

### Comments

However, if you add another member to `struct S` in `nested_value_types.cpp` (for example, `double d;`) and

recompile the component without also recompiling the client, the result is an unhandled exception (of type [System.IO.FileLoadException](#)).

## How to test for equality

In the following sample, a test for equality that uses Managed Extensions for C++ is based on what the handles refer to.

### Example

```
// mcppv2_equality_test.cpp
// compile with: /clr /LD
using namespace System;

bool Test1() {
    String ^ str1 = "test";
    String ^ str2 = "test";
    return (str1 == str2);
}
```

The IL for this program shows that the return value is implemented by using a call to `op_Equality`.

```
IL_0012:  call      bool [mscorlib]System.String::op_Equality(string, string)
```

## How to diagnose and fix assembly compatibility problems

When the version of an assembly referenced at compile time doesn't match the version of the assembly referenced at runtime, various problems may occur.

When an assembly is compiled, other assemblies may be referenced with the `#using` syntax. During the compilation, these assemblies are accessed by the compiler. Information from these assemblies is used to make optimization decisions.

However, if the referenced assembly is changed and recompiled, also recompile the referencing assembly that's dependent on it. Otherwise, the assemblies might become incompatible. Optimization decisions that were valid at first might not be correct for the new assembly version. Various runtime errors might occur because of these incompatibilities. There's no specific exception produced in such cases. The way the failure is reported at runtime depends on the nature of the code change that caused the problem.

These errors shouldn't be a problem in your final production code as long as the entire application is rebuilt for the released version of your product. Assemblies that are released to the public should be marked with an official version number, which will ensure that these problems are avoided. For more information, see [Assembly Versioning](#).

### To diagnose and fix an incompatibility error

You may encounter runtime exceptions or other error conditions in code that references another assembly. If you can't identify another cause, the problem may be an out of date assembly.

1. First, isolate and reproduce the exception or other error condition. A problem that occurs due to an outdated exception should be reproducible.
2. Check the timestamp of any assemblies referenced in your application.
3. If the timestamps of any referenced assemblies are later than the timestamp of your application's last compilation, then your application is out of date. If it's out of date, recompile your application with the most recent assemblies, and edit your code if necessary.

4. Rerun the application, perform the steps that reproduce the problem, and verify that the exception doesn't occur.

## Example

The following program illustrates the problem: it first reduces the accessibility of a method, and then tries to access that method in another assembly without recompiling. Compile `changeaccess.cpp` first. It's the referenced assembly that will change. Then compile `referencing.cpp`. It should successfully compile. Next, reduce the accessibility of the called method. Recompile `changeaccess.cpp` with the compiler option `/DCHANGE_ACCESS`. It makes the `access_me` method `protected`, rather than `public`, so it can't be called from outside `Test` or its derivatives. Without recompiling `referencing.exe`, rerun the application. A `MethodAccessException` occurs.

```
// changeaccess.cpp
// compile with: /clr:safe /LD
// After the initial compilation, add /DCHANGE_ACCESS and rerun
// referencing.exe to introduce an error at runtime. To correct
// the problem, recompile referencing.exe

public ref class Test {
#if defined(CHANGE_ACCESS)
protected:
#else
public:
#endif

    int access_me() {
        return 0;
    }
};

};
```

Here's the source for the referencing assembly:

```
// referencing.cpp
// compile with: /clr:safe
#using <changeaccess.dll>

// Force the function to be inline, to override the compiler's own
// algorithm.
__forceinline
int CallMethod(Test^ t) {
    // The call is allowed only if access_me is declared public
    return t->access_me();
}

int main() {
    Test^ t = gcnew Test();
    try
    {
        CallMethod(t);
        System::Console::WriteLine("No exception.");
    }
    catch (System::Exception ^ e)
    {
        System::Console::WriteLine("Exception!");
    }
    return 0;
}
```

#### See also

Interoperability with other .NET languages (C++/CLI)

Managed types (C++/CLI)

`#using` directive

# Reflection (C++/CLI)

9/20/2022 • 9 minutes to read • [Edit Online](#)

Reflection allows known data types to be inspected at runtime. Reflection allows the enumeration of data types in a given assembly, and the members of a given class or value type can be discovered. This is true regardless of whether the type was known or referenced at compile time. This makes reflection a useful feature for development and code management tools.

Note that the assembly name provided is the strong name (see [Creating and Using Strong-Named Assemblies](#)), which includes the assembly version, culture, and signing information. Note also that the name of the namespace in which the data type is defined can be retrieved, along with the name of the base class.

The most common way to access reflection features is through the `GetType` method. This method is provided by `System.Object`, from which all garbage-collected classes derive.

## NOTE

Reflection on an .exe built with the Microsoft C++ compiler is only allowed if the .exe is built with the `/clr:pure` or `/clr:safe` compiler options. The `/clr:pure` and `/clr:safe` compiler options are deprecated in Visual Studio 2015 and unavailable in Visual Studio 2017. See [/clr \(Common Language Runtime Compilation\)](#) for more information.

For more information, see [System.Reflection](#)

## Example: GetType

The `GetType` method returns a pointer to a `Type` class object, which describes the type upon which the object is based. (The `Type` object does not contain any instance-specific information.) One such item is the full name of the type, which can be displayed as follows:

Note that the type name includes the full scope in which the type is defined, including the namespace, and that it is displayed in .NET syntax, with a dot as the scope resolution operator.

```
// vcpp_reflection.cpp
// compile with: /clr
using namespace System;
int main() {
    String ^ s = "sample string";
    Console::WriteLine("full type name of '{0}' is '{1}'", s, s->GetType());
}
```

```
full type name of 'sample string' is 'System.String'
```

## Example: boxed value types

Value types can be used with the `GetType` function as well, but they must be boxed first.

```
// vcpp_reflection_2.cpp
// compile with: /clr
using namespace System;
int main() {
    Int32 i = 100;
    Object ^ o = i;
    Console::WriteLine("type of i = '{0}'", o->GetType());
}
```

```
type of i = 'System.Int32'
```

## Example: typeid

As with the `GetType` method, the `typeid` operator returns a pointer to a `Type` object, so this code indicates the type name `System.Int32`. Displaying type names is the most basic feature of reflection, but a potentially more useful technique is to inspect or discover the valid values for enumerated types. This can be done by using the static `Enum::GetNames` function, which returns an array of strings, each containing an enumeration value in text form. The following sample retrieves an array of strings that describes the value enumeration values for the `Options` (CLR) enum and displays them in a loop.

If a fourth option is added to the `Options` enumeration, this code will report the new option without recompilation, even if the enumeration is defined in a separate assembly.

```
// vcpp_reflection_3.cpp
// compile with: /clr
using namespace System;

enum class Options {    // not a native enum
    Option1, Option2, Option3
};

int main() {
    array<String^>^ names = Enum::GetNames(Options::typeid);

    Console::WriteLine("there are {0} options in enum '{1}'",
        names->Length, Options::typeid);

    for (int i = 0 ; i < names->Length ; i++)
        Console::WriteLine("{0}: {1}", i, names[i]);

    Options o = Options::Option2;
    Console::WriteLine("value of 'o' is {0}", o);
}
```

```
there are 3 options in enum 'Options'
0: Option1
1: Option2
2: Option3
value of 'o' is Option2
```

## Example: GetType members and properties

The `GetType` object supports a number of members and properties that can be used to examine a type. This code retrieves and displays some of this information:

```

// vcpp_reflection_4.cpp
// compile with: /clr
using namespace System;
int main() {
    Console::WriteLine("type information for 'String':");
    Type ^ t = String::typeid;

    String ^ assemblyName = t->Assembly->FullName;
    Console::WriteLine("assembly name: {0}", assemblyName);

    String ^ nameSpace = t->Namespace;
    Console::WriteLine("namespace: {0}", nameSpace);

    String ^ baseType = t->BaseType->FullName;
    Console::WriteLine("base type: {0}", baseType);

    bool isArray = t->IsArray;
    Console::WriteLine("is array: {0}", isArray);

    bool isClass = t->IsClass;
    Console::WriteLine("is class: {0}", isClass);
}

```

```

type information for 'String':
assembly name: mscorlib, Version=1.0.5000.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089
namespace: System
base type: System.Object
is array: False
is class: True

```

## Example: enumeration of types

Reflection also allows the enumeration of types within an assembly and the members within classes. To demonstrate this feature, define a simple class:

```

// vcpp_reflection_5.cpp
// compile with: /clr /LD
using namespace System;
public ref class TestClass {
    int m_i;
public:
    TestClass() {}
    void SimpleTestMember1() {}
    String ^ SimpleMember2(String ^ s) { return s; }
    int TestMember(int i) { return i; }
    property int Member {
        int get() { return m_i; }
        void set(int i) { m_i = i; }
    }
};

```

## Example: inspection of assemblies

If the code above is compiled into a DLL called vcpp\_reflection\_6.dll, you can then use reflection to inspect the contents of this assembly. This involves using the static reflection API function `xref:System.Reflection.Assembly.Load%2A?displayProperty=nameWithType` to load the assembly. This function returns the address of an `Assembly` object that can then be queried about the modules and types within.

Once the reflection system successfully loads the assembly, an array of **Type** objects is retrieved with the **Assembly::GetTypes** function. Each array element contains information about a different type, although in this case, only one class is defined. Using a loop, each **Type** in this array is queried about the type members using the **Type::GetMembers** function. This function returns an array of **MethodInfo** objects, each object containing information about the member function, data member, or property in the type.

Note that the list of methods includes the functions explicitly defined in **TestClass** and the functions implicitly inherited from the **System::Object** class. As part of being described in .NET rather than in Visual C++ syntax, properties appear as the underlying data member accessed by the get/set functions. The get/set functions appear in this list as regular methods. Reflection is supported through the common language runtime, not by the Microsoft C++ compiler.

Although you used this code to inspect an assembly that you defined, you can also use this code to inspect .NET assemblies. For example, if you change **TestAssembly** to **mscorlib**, then you will see a listing of every type and method defined in **mscorlib.dll**.

```
// vcpp_reflection_6.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;
using namespace System::Reflection;
int main() {
    Assembly ^ a = nullptr;
    try {
        // load assembly -- do not use file extension
        // will look for .dll extension first
        // then .exe with the filename
        a = Assembly::Load("vcpp_reflection_5");
    }
    catch (FileNotFoundException ^ e) {
        Console::WriteLine(e->Message);
        return -1;
    }

    Console::WriteLine("assembly info:");
    Console::WriteLine(a->FullName);
    array<Type^>^ typeArray = a->GetTypes();

    Console::WriteLine("type info ({0} types):", typeArray->Length);

    int totalTypes = 0;
    int totalMembers = 0;
    for (int i = 0 ; i < typeArray->Length ; i++) {
        // retrieve array of member descriptions
        array<MemberInfo^>^ member = typeArray[i]->GetMembers();

        Console::WriteLine("    members of {0} ({1} members):",
            typeArray[i]->FullName, member->Length);
        for (int j = 0 ; j < member->Length ; j++) {
            Console::Write("        ({0})",
                member[j]->MemberType.ToString() );
            Console::Write("{0} ", member[j]);
            Console::WriteLine("");
            totalMembers++;
        }
        totalTypes++;
    }
    Console::WriteLine("{0} total types, {1} total members",
        totalTypes, totalMembers);
}
```

## How to: Implement a Plug-In Component Architecture using

# Reflection

The following code examples demonstrate the use of reflection to implement a simple "plug-in" architecture. The first listing is the application, and the second is the plug-in. The application is a multiple document form that populates itself using any form-based classes found in the plug-in DLL provided as a command-line argument.

The application attempts to load the provided assembly using the [System.Reflection.Assembly.Load](#) method. If successful, the types inside the assembly are enumerated using the [System.Reflection.Assembly.GetTypes](#) method. Each type is then checked for compatibility using the [System.Type.IsAssignableFrom](#) method. In this example, classes found in the provided assembly must be derived from the [Form](#) class to qualify as a plug-in.

Compatible classes are then instantiated with the [System.Activator.CreateInstance](#) method, which accepts a [Type](#) as an argument and returns a pointer to a new instance. Each new instance is then attached to the form and displayed.

Note that the [Load](#) method does not accept assembly names that include the file extension. The main function in the application trims any provided extensions, so the following code example works in either case.

## Example app

The following code defines the application that accepts plug-ins. An assembly name must be provided as the first argument. This assembly should contain at least one public [Form](#) derived type.

```
// plugin_application.cpp
// compile with: /clr /c
#using <system.dll>
#using <system.drawing.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Reflection;

ref class PluggableForm : public Form {
public:
    PluggableForm() {}
    PluggableForm(Assembly^ plugAssembly) {
        Text = "plug-in example";
        Size = Drawing::Size(400, 400);
        IsMdiContainer = true;

        array<Type^>^ types = plugAssembly->GetTypes();
        Type^ formType = Form::typeid;

        for (int i = 0 ; i < types->Length ; i++) {
            if (formType->IsAssignableFrom(types[i])) {
                // Create an instance given the type description.
                Form^ f = dynamic_cast<Form^> (Activator::CreateInstance(types[i]));
                if (f) {
                    f->Text = types[i]->ToString();
                    f->MdiParent = this;
                    f->>Show();
                }
            }
        }
    }
};

int main() {
    Assembly^ a = Assembly::LoadFrom("plugin_application.exe");
    Application::Run(gcnew PluggableForm(a));
}
```

## Example plug-ins

The following code defines three classes derived from [Form](#). When the name of the resulting assembly name is passed to the executable in the previous listing, each of these three classes will be discovered and instantiated, despite the fact that they were all unknown to the hosting application at compile time.

```
// plugin_assembly.cpp
// compile with: /clr /LD
#using <system.dll>
#using <system.drawing.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Reflection;
using namespace System::Drawing;

public ref class BlueForm : public Form {
public:
    BlueForm() {
        BackColor = Color::Blue;
    }
};

public ref class CircleForm : public Form {
protected:
    virtual void OnPaint(PaintEventArgs^ args) override {
        args->Graphics->FillEllipse(Brushes::Green, ClientRectangle);
    }
};

public ref class StarburstForm : public Form {
public:
    StarburstForm(){
        BackColor = Color::Black;
    }
protected:
    virtual void OnPaint(PaintEventArgs^ args) override {
        Pen^ p = gcnew Pen(Color::Red, 2);
        Random^ r = gcnew Random( );
        Int32 w = ClientSize.Width;
        Int32 h = ClientSize.Height;
        for (int i=0; i<100; i++) {
            float x1 = w / 2;
            float y1 = h / 2;
            float x2 = r->Next(w);
            float y2 = r->Next(h);
            args->Graphics->DrawLine(p, x1, y1, x2, y2);
        }
    }
};
}
```

## How to: Enumerate Data Types in Assemblies using Reflection

The following code demonstrates the enumeration of public types and members using [System.Reflection](#).

Given the name of an assembly, either in the local directory or in the GAC, the code below attempts to open the assembly and retrieve descriptions. If successful, each type is displayed with its public members.

Note that [System.Reflection.Assembly.Load](#) requires that no file extension is used. Therefore, using "mscorlib.dll" as a command-line argument will fail, while using just "mscorlib" will result the display of the .NET Framework types. If no assembly name is provided, the code will detect and report the types within the current assembly (the EXE resulting from this code).

## Example

```
// self_reflection.cpp
// compile with: /clr
using namespace System;
using namespace System::Reflection;
using namespace System::Collections;

public ref class ExampleType {
public:
    ExampleType() {}
    void Func() {}
};

int main() {
    String^ delimStr = " ";
    array<Char>^ delimiter = delimStr->ToCharArray( );
    array<String>^ args = Environment::CommandLine->Split( delimiter );

// replace "self_reflection.exe" with an assembly from either the local
// directory or the GAC
    Assembly^ a = Assembly::LoadFrom("self_reflection.exe");
    Console::WriteLine(a);

    int count = 0;
    array<Type>^ types = a->GetTypes();
    IEnumerator^ typeIter = types->GetEnumerator();

    while ( typeIter->MoveNext() ) {
        Type^ t = dynamic_cast<Type>(typeIter->Current);
        Console::WriteLine(" {0}", t->ToString());

        array<MemberInfo>^ members = t->GetMembers();
        IEnumerator^ memberIter = members->GetEnumerator();
        while ( memberIter->MoveNext() ) {
            MemberInfo^ mi = dynamic_cast<MemberInfo>(memberIter->Current);
            Console::Write(" {0}", mi->ToString( ));
            if (mi->MemberType == MemberTypes::Constructor)
                Console::Write(" (constructor)");

            Console::WriteLine();
        }
        count++;
    }
    Console::WriteLine("{0} types found", count);
}
```

## See also

- [.NET Programming with C++/CLI \(Visual C++\)](#)

# Strong Name Assemblies (Assembly Signing) (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic discusses how you can sign your assembly, often referred to as giving your assembly a strong name.

## Remarks

When using Visual C++, use linker options to sign your assembly to avoid issues related to the CLR attributes for assembly signing:

- [AssemblyDelaySignAttribute](#)
- [AssemblyKeyFileAttribute](#)
- [AssemblyKeyNameAttribute](#)

Reasons for not using the attributes include the fact that the key name is visible in assembly metadata, which can be a security risk if the file name includes confidential information. Also, the build process used by the Visual C++ development environment will invalidate the key with which the assembly is signed if you use CLR attributes to give an assembly a strong name, and then run a post-processing tool like mt.exe on the assembly.

If you build at the command line, use linker options to sign your assembly, and then run a post-processing tool (like mt.exe), you will need to re-sign the assembly with sn.exe. Alternatively, you can build and delay sign the assembly and after running post-processing tools, complete the signing.

If you use the signing attributes when building in the development environment, you can successfully sign the assembly by explicitly calling sn.exe ([Sn.exe \(Strong Name Tool\)](#)) in a post-build event. For more information, see [Specifying Build Events](#). Build times may be less if you use attributes and a post-build event, compared to using a linker options.

The following linker options support assembly signing:

- [/DELAYSIGN \(Partially Sign an Assembly\)](#)
- [/KEYFILE \(Specify Key or Key Pair to Sign an Assembly\)](#)
- [/KEYCONTAINER \(Specify a Key Container to Sign an Assembly\)](#)

For more information on strong assemblies, see [Creating and Using Strong-Named Assemblies](#).

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

# Debug Class (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

When using [Debug](#) in a Visual C++ application, the behavior does not change between a debug and a release build.

## Remarks

The behavior for [Trace](#) is identical to the behavior for the Debug class, but is dependent on the symbol TRACE being defined. This means that you must `#ifdef` any Trace-related code to prevent debug behavior in a release build.

## Example: Always executes output statements

### Description

The following sample always executes the output statements, regardless of whether you compile with `/DDEBUG` or `/DTRACE`.

### Code

```
// mcpp_debug_class.cpp
// compile with: /clr
#using <system.dll>
using namespace System::Diagnostics;
using namespace System;

int main() {
    Trace::Listeners->Add( gcnew TextWriterTraceListener( Console::Out ) );
    Trace::AutoFlush = true;
    Trace::Indent();
    Trace::WriteLine( "Entering Main" );
    Console::WriteLine( "Hello World." );
    Trace::WriteLine( "Exiting Main" );
    Trace::Unindent();

    Debug::WriteLine("test");
}
```

### Output

```
Entering Main
Hello World.
Exiting Main
test
```

## Example: Use `#ifdef` and `#endif` directives

### Description

To get the expected behavior (that is, no "test" output printed for a release build), you must use the `#ifdef` and `#endif` directives. The previous code sample is modified below to demonstrate this fix:

### Code

```
// mcpp_debug_class2.cpp
// compile with: /clr
#using <system.dll>
using namespace System::Diagnostics;
using namespace System;

int main() {
    Trace::Listeners->Add( gcnew TextWriterTraceListener( Console::Out ) );
    Trace::AutoFlush = true;
    Trace::Indent();

#ifndef TRACE // checks for a debug build
    Trace::WriteLine( "Entering Main" );
    Console::WriteLine( "Hello World." );
    Trace::WriteLine( "Exiting Main" );
#endif
    Trace::Unindent();

#ifndef DEBUG // checks for a debug build
    Debug::WriteLine("test");
#endif //ends the conditional block
}
```

## See also

[.NET Programming with C++/CLI \(Visual C++\)](#)

# STL/CLR Library Reference

9/20/2022 • 2 minutes to read • [Edit Online](#)

The STL/CLR Library provides an interface similar to the C++ Standard Library containers for use with C++ and the .NET Framework common language runtime (CLR). STL/CLR is completely separate from the Microsoft implementation of the C++ Standard Library. STL/CLR is maintained for legacy support but is not kept up-to-date with the C++ standard. We strongly recommend using the native [C++ Standard Library](#) containers instead of STL/CLR whenever possible.

To use STL/CLR:

- Include headers from the `cliext` include subdirectory instead of the usual C++ Standard Library equivalents.
- Qualify library names with `cliext::` instead of `std::`.

The STL/CLR Library provides an STL-like interface for use with C++ and the .NET Framework common language runtime (CLR). This library is maintained for legacy support but is not kept up-to-date with the C++ standard. We strongly recommend using the native [C++ Standard Library](#) containers instead of STL/CLR.

## In This Section

### [cliext Namespace](#)

Discusses the namespace that contains all the types of the STL/CLR Library.

### [STL/CLR Containers](#)

Provides an overview of the containers that are found in the C++ Standard Library, including requirements for container elements, types of elements that can be inserted, and ownership issues.

### [Requirements for STL/CLR Container Elements](#)

Describes minimum requirements for all reference types that are inserted into C++ Standard Library containers.

### [How to: Convert from a .NET Collection to a STL/CLR Container](#)

Describes how to convert a .NET collection to an STL/CLR container.

### [How to: Convert from a STL/CLR Container to a .NET Collection](#)

Describes how to convert an STL/CLR container to a .NET collection.

### [How to: Expose an STL/CLR Container from an Assembly](#)

Shows how to display the elements of several STL/CLR containers written in a C++ assembly.

In addition, this section also describes the following components of STL/CLR:

`adapter` (STL/CLR)  
`algorithm` (STL/CLR)  
`deque` (STL/CLR)  
`for each , in`  
`functional` (STL/CLR)  
`hash_map` (STL/CLR)  
`hash_multimap` (STL/CLR)  
`hash_multiset` (STL/CLR)  
`hash_set` (STL/CLR)  
`list` (STL/CLR)\

[map](#) (STL/CLR)  
[multimap](#) (STL/CLR)  
[multiset](#) (STL/CLR)  
[numeric](#) (STL/CLR)  
[priority\\_queue](#) (STL/CLR)  
[queue](#) (STL/CLR)  
[set](#) (STL/CLR)  
[stack](#) (STL/CLR)  
[utility](#) (STL/CLR)  
[vector](#) (STL/CLR)\\

## See also

[C++ Standard Library](#)

# cliext Namespace

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `cliext` namespace contains all the types of the STL/CLR library. For a list of all these types, and links to more information on the STL/CLR types, see [STL/CLR Library Reference](#).

## See also

[STL/CLR Library Reference](#)

# STL/CLR Containers

9/20/2022 • 6 minutes to read • [Edit Online](#)

The STL/CLR Library consists of containers that are similar to those found in the C++ Standard Library, but it runs within the managed environment of the .NET Framework. It is not kept up-to-date with the actual C++ Standard Library and is maintained for legacy support.

This document provides an overview of the containers in STL/CLR, such as the requirements for container elements, the types of elements that you can insert into the containers, and ownership issues with the elements in the containers. Where appropriate, differences between the native C++ Standard Library and STL/CLR are mentioned.

## Requirements for Container Elements

All elements inserted into STL/CLR containers must obey certain guidelines. For more information, see [Requirements for STL/CLR Container Elements](#).

## Valid Container Elements

STL/CLR containers can hold one of two types of elements:

- Handles to reference types.
- Reference types.
- Unboxed value types.

You cannot insert boxed value types into any of the STL/CLR containers.

### Handles to Reference Types

You can insert a handle to a reference type into an STL/CLR container. A handle in C++ that targets the CLR is analogous to a pointer in native C++. For more information, see [Handle to Object Operator \(^\)](#).

#### Example

The following example shows how to insert a handle to an Employee object into a `cliext::set`.

```
// cliext_container_valid_reference_handle.cpp
// compile with: /clr

#include <cliext/set>

using namespace cliext;
using namespace System;

ref class Employee
{
public:
    // STL/CLR containers might require a public constructor, so it
    // is a good idea to define one.
    Employee() :
        name(nullptr),
        employeeNumber(0) { }

    // All STL/CLR containers require a public copy constructor.
    Employee(const Employee% orig) :
        name(orig.name),
        employeeNumber(orig.employeeNumber) { }
```

```

// All STL/CLR containers require a public assignment operator.
Employee% operator=(const Employee% orig)
{
    if (this != %orig)
    {
        name = orig.name;
        employeeNumber = orig.employeeNumber;
    }

    return *this;
}

// All STL/CLR containers require a public destructor.
~Employee() { }

// Associative containers such as maps and sets
// require a comparison operator to be defined
// to determine proper ordering.
bool operator<(const Employee^ rhs)
{
    return (employeeNumber < rhs->employeeNumber);
}

// The employee's name.
property String^ Name
{
    String^ get() { return name; }
    void set(String^ value) { name = value; }
}

// The employee's employee number.
property int EmployeeNumber
{
    int get() { return employeeNumber; }
    void set(int value) { employeeNumber = value; }
}

private:
    String^ name;
    int employeeNumber;
};

int main()
{
    // Create a new employee object.
    Employee^ empl1419 = gcnew Employee();
    empl1419->Name = L"Darin Lockert";
    empl1419->EmployeeNumber = 1419;

    // Add the employee to the set of all employees.
    set<Employee^>^ emplSet = gcnew set<Employee^>();
    emplSet->insert(empl1419);

    // List all employees of the company.
    for each (Employee^ empl in emplSet)
    {
        Console::WriteLine("Employee Number {0}: {1}",
                           empl->EmployeeNumber, empl->Name);
    }

    return 0;
}

```

## Reference Types

It is also possible to insert a reference type (rather than a handle to a reference type) into a STL/CLR container.

The main difference here is that when a container of reference types is deleted, the destructor is called for all elements inside that container. In a container of handles to reference types, the destructors for these elements would not be called.

#### Example

The following example shows how to insert an Employee object into a `cliext::set`.

```
// cliext_container_valid_reference.cpp
// compile with: /clr

#include <cliext/set>

using namespace cliext;
using namespace System;

ref class Employee
{
public:
    // STL/CLR containers might require a public constructor, so it
    // is a good idea to define one.
    Employee() :
        name(nullptr),
        employeeNumber(0) { }

    // All STL/CLR containers require a public copy constructor.
    Employee(const Employee% orig) :
        name(orig.name),
        employeeNumber(orig.employeeNumber) { }

    // All STL/CLR containers require a public assignment operator.
    Employee% operator=(const Employee% orig)
    {
        if (this != %orig)
        {
            name = orig.name;
            employeeNumber = orig.employeeNumber;
        }

        return *this;
    }

    // All STL/CLR containers require a public destructor.
    ~Employee() { }

    // Associative containers such as maps and sets
    // require a comparison operator to be defined
    // to determine proper ordering.
    bool operator<(const Employee^ rhs)
    {
        return (employeeNumber < rhs->employeeNumber);
    }

    // The employee's name.
    property String^ Name
    {
        String^ get() { return name; }
        void set(String^ value) { name = value; }
    }

    // The employee's employee number.
    property int EmployeeNumber
    {
        int get() { return employeeNumber; }
        void set(int value) { employeeNumber = value; }
    }

private:
```

```

    String^ name;
    int employeeNumber;
};

int main()
{
    // Create a new employee object.
    Employee empl1419;
    empl1419.Name = L"Darin Lockert";
    empl1419.EmployeeNumber = 1419;

    // Add the employee to the set of all employees.
    set<Employee>^ emplSet = gcnew set<Employee>();
    emplSet->insert(empl1419);

    // List all employees of the company.
    for each (Employee^ empl in emplSet)
    {
        Console::WriteLine("Employee Number {0}: {1}",
            empl->EmployeeNumber, empl->Name);
    }

    return 0;
}

```

## Unboxed Value Types

You can also insert an unboxed value type into an STL/CLR container. An unboxed value type is a value type that has not been *boxed* into a reference type.

A value type element can be one of the standard value types, such as an `int`, or it can be a user-defined value type, such as a `value class`. For more information, see [Classes and Structs](#)

### Example

The following example modifies the first example by making the `Employee` class a value type. This value type is then inserted into a `cliext::set` just as in the first example.

```

// cliext_container_valid_valuetype.cpp
// compile with: /clr

#include <cliext/set>

using namespace cliext;
using namespace System;

value class Employee
{
public:
    // Associative containers such as maps and sets
    // require a comparison operator to be defined
    // to determine proper ordering.
    bool operator<(const Employee^ rhs)
    {
        return (employeeNumber < rhs->employeeNumber);
    }

    // The employee's name.
    property String^ Name
    {
        String^ get() { return name; }
        void set(String^ value) { name = value; }
    }

    // The employee's employee number.
    property int EmployeeNumber
    {
        int get() { return employeeNumber; }
        void set(int value) { employeeNumber = value; }
    }

private:
    String^ name;
    int employeeNumber;
};

int main()
{
    // Create a new employee object.
    Employee empl1419;
    empl1419.Name = L"Darin Lockert";
    empl1419.EmployeeNumber = 1419;

    // Add the employee to the set of all employees.
    set<Employee>^ emplSet = gcnew set<Employee>();
    emplSet->insert(empl1419);

    // List all employees of the company.
    for each (Employee empl in emplSet)
    {
        Console::WriteLine("Employee Number {0}: {1}",
                           empl.EmployeeNumber, empl.Name);
    }

    return 0;
}

```

If you attempt to insert a handle to a value type into a container, [Compiler Error C3225](#) is generated.

### Performance and Memory Implications

You must consider several factors when determining whether to use handles to reference types or value types as container elements. If you decide to use value types, remember that a copy of the element is made every time an element is inserted into the container. For small objects, this should not be a problem, but if the objects being

inserted are large, performance might suffer. Also, if you are using value types, it is impossible to store one element in multiple containers at the same time because each container would have its own copy of the element.

If you decide to use handles to reference types instead, performance might increase because it is not necessary to make a copy of the element when it is inserted in the container. Also, unlike with value types, the same element can exist in multiple containers. However, if you decide to use handles, you must take care to ensure that the handle is valid and that the object it refers to has not been deleted elsewhere in the program.

## Ownership Issues with Containers

Containers in STL/CLR work on value semantics. Every time you insert an element into a container, a copy of that element is inserted. If you want to get reference-like semantics, you can insert a handle to an object rather than the object itself.

When you call the clear or erase method of a container of handle objects, the objects that the handles refer to are not freed from memory. You must either explicitly delete the object, or, because these objects reside on the managed heap, allow the garbage collector to free the memory once it determines that the object is no longer being used.

## See also

[C++ Standard Library Reference](#)

# Requirements for STL/CLR Container Elements

9/20/2022 • 2 minutes to read • [Edit Online](#)

All reference types that are inserted into STL/CLR containers must have, at a minimum, the following elements:

- A public copy constructor.
- A public assignment operator.
- A public destructor.

Furthermore, associative containers such as `set` and `map` must have a public comparison operator defined, which is `operator<` by default. Some operations on containers might also require a public default constructor and a public equivalence operator to be defined.

Like reference types, value types and handles to reference types that are to be inserted into an associative container must have a comparison operator such as `operator<` defined. The requirements for a public copy constructor, public assignment operator, and a public destructor do not exist for value types or handles to reference types.

## See also

[C++ Standard Library Reference](#)

# How to: Convert from a .NET Collection to a STL/CLR Container

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic shows how to convert .NET collections to their equivalent STL/CLR containers. As an example we show how to convert a .NET `List<T>` to a STL/CLR `vector` and how to convert a .NET `Dictionary< TKey, TValue >` to a STL/CLR `map`, but the procedure is similar for all collections and containers.

## To create a container from a collection

1. To convert an entire collection, create a STL/CLR container and pass the collection to the constructor.

The first example demonstrates this procedure.

-OR-

1. Create a generic STL/CLR container by creating a `collection_adapter` object. This template class takes a .NET collection interface as an argument. To verify which interfaces are supported, see [collection\\_adapter \(STL/CLR\)](#).
2. Copy the contents of the .NET collection to the container. This can be done by using a STL/CLR `algorithm`, or by iterating over the .NET collection and inserting a copy of each element into the STL/CLR container.

The second example demonstrates this procedure.

## Examples

In this example, we create a generic `List<T>` and add 5 elements to it. Then, we create a `vector` using the constructor that takes a `IEnumerable<T>` as an argument.

```

// cliext_convert_list_to_vector.cpp
// compile with: /clr

#include <cliext/adapters>
#include <cliext/algorithms>
#include <cliext/vector>

using namespace System;
using namespace System::Collections;
using namespace System::Collections::Generic;

int main(array<System::String ^> ^args)
{
    List<int> ^primeNumbersColl = gcnew List<int>();
    primeNumbersColl->Add(2);
    primeNumbersColl->Add(3);
    primeNumbersColl->Add(5);
    primeNumbersColl->Add(7);
    primeNumbersColl->Add(11);

    cliext::vector<int> ^primeNumbersCont =
        gcnew cliext::vector<int>(primeNumbersColl);

    Console::WriteLine("The contents of the cliext::vector are:");
    cliext::vector<int>::const_iterator it;
    for (it = primeNumbersCont->begin(); it != primeNumbersCont->end(); it++)
    {
        Console::WriteLine(*it);
    }
}

```

```

The contents of the cliext::vector are:
2
3
5
7
11

```

In this example, we create a generic `Dictionary< TKey, TValue >` and add 5 elements to it. Then, we create a `collection_adapter` to wrap the `Dictionary< TKey, TValue >` as a simple STL/CLR container. Finally, we create a `map` and copy the contents of the `Dictionary< TKey, TValue >` to the `map` by iterating over the `collection_adapter`. During this process, we create a new pair by using the `make_pair` function, and insert the new pair directly into the `map`.

```

// cliext_convert_dictionary_to_map.cpp
// compile with: /clr

#include <cliext/adapter>
#include <cliext/algorithm>
#include <cliext/map>

using namespace System;
using namespace System::Collections;
using namespace System::Collections::Generic;

int main(array<System::String ^> ^args)
{
    System::Collections::Generic::Dictionary<float, int> ^dict =
        gcnew System::Collections::Generic::Dictionary<float, int>();
    dict->Add(42.0, 42);
    dict->Add(13.0, 13);
    dict->Add(74.0, 74);
    dict->Add(22.0, 22);
    dict->Add(0.0, 0);

    cliext::collection_adapter<System::Collections::Generic::IDictionary<float, int>> dictAdapter(dict);
    cliext::map<float, int> aMap;
    for each (KeyValuePair<float, int> ^kvp in dictAdapter)
    {
        cliext::pair<float, int> aPair = cliext::make_pair(kvp->Key, kvp->Value);
        aMap.insert(aPair);
    }

    Console::WriteLine("The contents of the cliext::map are:");
    cliext::map<float, int>::const_iterator it;
    for (it = aMap.begin(); it != aMap.end(); it++)
    {
        Console::WriteLine("Key: {0:F} Value: {1}", it->first, it->second);
    }
}

```

```

The contents of the cliext::map are:
Key: 0.00 Value: 0
Key: 13.00 Value: 13
Key: 22.00 Value: 22
Key: 42.00 Value: 42
Key: 74.00 Value: 74

```

## See also

- [STL/CLR Library Reference](#)
- [adapter \(STL/CLR\)](#)
- [How to: Convert from a STL/CLR Container to a .NET Collection](#)

# How to: Convert from a STL/CLR Container to a .NET Collection

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic shows how to convert STL/CLR containers to their equivalent .NET collections. As an example, we show how to convert a STL/CLR `vector` to a .NET `ICollection<T>` and how to convert a STL/CLR `map` to a .NET `IDictionary< TKey, TValue >`, but the procedure is similar for all collections and containers.

## To create a collection from a container

1. Use one of the following methods:

- To convert part of a container, call the `make_collection` function, and pass the begin iterator and end iterator of the STL/CLR container to be copied into the .NET collection. This template function takes an STL/CLR iterator as a template argument. The first example demonstrates this method.
- To convert an entire container, cast the container to an appropriate .NET collection interface or interface collection. The second example demonstrates this method.

## Examples

In this example, we create a STL/CLR `vector` and add 5 elements to it. Then, we create a .NET collection by calling the `make_collection` function. Finally, we display the contents of the newly created collection.

```
// cliext_convert_vector_to_icollection.cpp
// compile with: /clr

#include <cliext/adapter>
#include <cliext/vector>

using namespace cliext;
using namespace System;
using namespace System::Collections::Generic;

int main(array<System::String ^> ^args)
{
    cliext::vector<int> primeNumbersCont;
    primeNumbersCont.push_back(2);
    primeNumbersCont.push_back(3);
    primeNumbersCont.push_back(5);
    primeNumbersCont.push_back(7);
    primeNumbersCont.push_back(11);

    System::Collections::Generic::ICollection<int> ^iColl =
        make_collection<cliext::vector<int>::iterator>(
            primeNumbersCont.begin() + 1,
            primeNumbersCont.end() - 1);

    Console::WriteLine("The contents of the System::Collections::Generic::ICollection are:");
    for each (int i in iColl)
    {
        Console::WriteLine(i);
    }
}
```

The contents of the System::Collections::Generic::ICollection are:

3  
5  
7

In this example, we create a STL/CLR `map` and add 5 elements to it. Then, we create a .NET `IDictionary<TKey,TValue>` and assign the `map` directly to it. Finally, we display the contents of the newly created collection.

```
// cliext_convert_map_to_idictionary.cpp
// compile with: /clr

#include <cliext/adapter>
#include <cliext/map>

using namespace cliext;
using namespace System;
using namespace System::Collections::Generic;

int main(array<System::String ^> ^args)
{
    cliext::map<float, int> ^aMap = gcnew cliext::map<float, int>;
    aMap->insert(cliext::make_pair<float, int>(42.0, 42));
    aMap->insert(cliext::make_pair<float, int>(13.0, 13));
    aMap->insert(cliext::make_pair<float, int>(74.0, 74));
    aMap->insert(cliext::make_pair<float, int>(22.0, 22));
    aMap->insert(cliext::make_pair<float, int>(0.0, 0));

    System::Collections::Generic::IDictionary<float, int> ^iDict = aMap;

    Console::WriteLine("The contents of the IDictionary are:");
    for each (KeyValuePair<float, int> ^kvp in iDict)
    {
        Console::WriteLine("Key: {0:F} Value: {1}", kvp->Key, kvp->Value);
    }
}
```

The contents of the IDictionary are:

Key: 0.00 Value: 0  
Key: 13.00 Value: 13  
Key: 22.00 Value: 22  
Key: 42.00 Value: 42  
Key: 74.00 Value: 74

## See also

[STL/CLR Library Reference](#)

[How to: Convert from a .NET Collection to a STL/CLR Container](#)

[range\\_adapter \(STL/CLR\)](#)

# How to: Expose an STL/CLR Container from an Assembly

9/20/2022 • 5 minutes to read • [Edit Online](#)

STL/CLR containers such as `list` and `map` are implemented as template ref classes. Because C++ templates are instantiated at compile time, two template classes that have exactly the same signature but are in different assemblies are actually different types. This means that template classes cannot be used across assembly boundaries.

To make cross-assembly sharing possible, STL/CLR containers implement the generic interface `ICollection<T>`. By using this generic interface, all languages that support generics, including C++, C#, and Visual Basic, can access STL/CLR containers.

This topic shows you how to display the elements of several STL/CLR containers written in a C++ assembly named `StlClrClassLibrary`. We show two assemblies to access `StlClrClassLibrary`. The first assembly is written in C++, and the second in C#.

If both assemblies are written in C++, you can access the generic interface of a container by using its `generic_container` typedef. For example, if you have a container of type `cliext::vector<int>`, then its generic interface is: `cliext::vector<int>::generic_container`. Similarly, you can get an iterator over the generic interface by using the `generic_iterator` typedef, as in: `cliext::vector<int>::generic_iterator`.

Since these typedefs are declared in C++ header files, assemblies written in other languages cannot use them. Therefore, to access the generic interface for `cliext::vector<int>` in C# or any other .NET language, use `System.Collections.Generic.ICollection<int>`. To iterate over this collection, use a `foreach` loop.

The following table lists the generic interface that each STL/CLR container implements:

STL/CLR CONTAINER	GENERIC INTERFACE
<code>deque&lt;T&gt;</code>	<code>ICollection&lt;T&gt;</code>
<code>hash_map&lt;K, V&gt;</code>	<code>IDictionary&lt;K, V&gt;</code>
<code>hash_multimap&lt;K, V&gt;</code>	<code>IDictionary&lt;K, V&gt;</code>
<code>hash_multiset&lt;T&gt;</code>	<code>ICollection&lt;T&gt;</code>
<code>hash_set&lt;T&gt;</code>	<code>ICollection&lt;T&gt;</code>
<code>list&lt;T&gt;</code>	<code>ICollection&lt;T&gt;</code>
<code>map&lt;K, V&gt;</code>	<code>IDictionary&lt;K, V&gt;</code>
<code>multimap&lt;K, V&gt;</code>	<code>IDictionary&lt;K, V&gt;</code>
<code>multiset&lt;T&gt;</code>	<code>ICollection&lt;T&gt;</code>
<code>set&lt;T&gt;</code>	<code>ICollection&lt;T&gt;</code>

`vector<T>``IICollection<T>`**NOTE**

Because the `queue`, `priority_queue`, and `stack` containers do not support iterators, they do not implement generic interfaces and cannot be accessed cross-assembly.

## Example 1

### Description

In this example, we declare a C++ class that contains private STL/CLR member data. We then declare public methods to grant access to the private collections of the class. We do it in two different ways, one for C++ clients and one for other .NET clients.

### Code

```

// StlClrClassLibrary.h
#pragma once

#include <cliext/deque>
#include <cliext/list>
#include <cliext/map>
#include <cliext/set>
#include <cliext/stack>
#include <cliext/vector>

using namespace System;
using namespace System::Collections::Generic;
using namespace cliext;

namespace StlClrClassLibrary {

public ref class StlClrClass
{
public:
    StlClrClass();

    // These methods can be called by a C++ class
    // in another assembly to get access to the
    // private STL/CLR types defined below.
    deque<wchar_t>::generic_container ^GetDequeCpp();
    list<float>::generic_container ^GetListCpp();
    map<int, String ^>::generic_container ^GetMapCpp();
    set<double>::generic_container ^GetSetCpp();
    vector<int>::generic_container ^GetVectorCpp();

    // These methods can be called by a non-C++ class
    // in another assembly to get access to the
    // private STL/CLR types defined below.
    ICollection<wchar_t> ^GetDequeCs();
    ICollection<float> ^GetListCs();
    IDictionary<int, String ^> ^GetMapCs();
    ICollection<double> ^GetSetCs();
    ICollection<int> ^GetVectorCs();

private:
    deque<wchar_t> ^aDeque;
    list<float> ^aList;
    map<int, String ^> ^aMap;
    set<double> ^aSet;
    vector<int> ^aVector;
};

}

```

## Example 2

### Description

In this example, we implement the class declared in Example 1. In order for clients to use this class library, we use the manifest tool **mt.exe** to embed the manifest file into the DLL. For details, see the code comments.

For more information on the manifest tool and side-by-side assemblies, see [Building C/C++ Isolated Applications and Side-by-side Assemblies](#).

### Code

```

// StlClrClassLibrary.cpp
// compile with: /clr /LD /link /manifest
// post-build command: (attrib -r StlClrClassLibrary.dll & mt /manifest StlClrClassLibrary.dll.manifest
// /outputresource:StlClrClassLibrary.dll:#2 & attrib +r StlClrClassLibrary.dll)

```

```

#include "StlClrClassLibrary.h"

namespace StlClrClassLibrary
{
    StlClrClass::StlClrClass()
    {
        aDeque = gcnew deque<wchar_t>();
        aDeque->push_back(L'a');
        aDeque->push_back(L'b');

        aList = gcnew list<float>();
        aList->push_back(3.14159f);
        aList->push_back(2.71828f);

        aMap = gcnew map<int, String ^>();
        aMap[0] = "Hello";
        aMap[1] = "World";

        aSet = gcnew set<double>();
        aSet->insert(3.14159);
        aSet->insert(2.71828);

        aVector = gcnew vector<int>();
        aVector->push_back(10);
        aVector->push_back(20);
    }

    deque<wchar_t>::generic_container ^StlClrClass::GetDequeCpp()
    {
        return aDeque;
    }

    list<float>::generic_container ^StlClrClass::GetListCpp()
    {
        return aList;
    }

    map<int, String ^>::generic_container ^StlClrClass::GetMapCpp()
    {
        return aMap;
    }

    set<double>::generic_container ^StlClrClass::GetSetCpp()
    {
        return aSet;
    }

    vector<int>::generic_container ^StlClrClass::GetVectorCpp()
    {
        return aVector;
    }

    ICollection<wchar_t> ^StlClrClass::GetDequeCs()
    {
        return aDeque;
    }

    ICollection<float> ^StlClrClass::GetListCs()
    {
        return aList;
    }

    IDictionary<int, String ^> ^StlClrClass::GetMapCs()
    {
        return aMap;
    }

    ICollection<double> ^StlClrClass::GetSetCs()
    {

```

```
        return aSet;
    }

ICollection<int> ^StlClrClass::GetVectorCs()
{
    return aVector;
}
}
```

## Example 3

### Description

In this example, we create a C++ client that uses the class library created in Examples 1 and 2. This client uses the `generic_container` typedefs of the STL/CLR containers to iterate over the containers and to display their contents.

### Code

```

// CppConsoleApp.cpp
// compile with: /clr /FUStlClrClassLibrary.dll

#include <cliext/deque>
#include <cliext/list>
#include <cliext/map>
#include <cliext/set>
#include <cliext/vector>

using namespace System;
using namespace StlClrClassLibrary;
using namespace cliext;

int main(array<System::String ^> ^args)
{
    StlClrClass theClass;

    Console::WriteLine("cliext::deque contents:");
    deque<wchar_t>::generic_container ^aDeque = theClass.GetDequeCpp();
    for each (wchar_t wc in aDeque)
    {
        Console::WriteLine(wc);
    }
    Console::WriteLine();

    Console::WriteLine("cliext::list contents:");
    list<float>::generic_container ^aList = theClass.GetListCpp();
    for each (float f in aList)
    {
        Console::WriteLine(f);
    }
    Console::WriteLine();

    Console::WriteLine("cliext::map contents:");
    map<int, String ^>::generic_container ^aMap = theClass.GetMapCpp();
    for each (map<int, String ^>::value_type rp in aMap)
    {
        Console::WriteLine("{0} {1}", rp->first, rp->second);
    }
    Console::WriteLine();

    Console::WriteLine("cliext::set contents:");
    set<double>::generic_container ^aSet = theClass.GetSetCpp();
    for each (double d in aSet)
    {
        Console::WriteLine(d);
    }
    Console::WriteLine();

    Console::WriteLine("cliext::vector contents:");
    vector<int>::generic_container ^aVector = theClass.GetVectorCpp();
    for each (int i in aVector)
    {
        Console::WriteLine(i);
    }
    Console::WriteLine();

    return 0;
}

```

## Output

```
cliext::deque contents:  
a  
b  
  
cliext::list contents:  
3.14159  
2.71828  
  
cliext::map contents:  
0 Hello  
1 World  
  
cliext::set contents:  
2.71828  
3.14159  
  
cliext::vector contents:  
10  
20
```

## Example 4

### Description

In this example, we create a C# client that uses the class library created in Examples 1 and 2. This client uses the [ICollection<T>](#) methods of the STL/CLR containers to iterate over the containers and to display their contents.

### Code

```

// CsConsoleApp.cs
// compile with: /r:Microsoft.VisualBasic.dll /r:StlClrClassLibrary.dll /r:System.dll

using System;
using System.Collections.Generic;
using StlClrClassLibrary;
using cliext;

namespace CsConsoleApp
{
    class Program
    {
        static int Main(string[] args)
        {
            StlClrClass theClass = new StlClrClass();

            Console.WriteLine("cliext::deque contents:");
            ICollection<char> iCollChar = theClass.GetDequeCs();
            foreach (char c in iCollChar)
            {
                Console.WriteLine(c);
            }
            Console.WriteLine();

            Console.WriteLine("cliext::list contents:");
            ICollection<float> iCollFloat = theClass.GetListCs();
            foreach (float f in iCollFloat)
            {
                Console.WriteLine(f);
            }
            Console.WriteLine();

            Console.WriteLine("cliext::map contents:");
            IDictionary<int, string> iDict = theClass.GetMapCs();
            foreach (KeyValuePair<int, string> kvp in iDict)
            {
                Console.WriteLine("{0} {1}", kvp.Key, kvp.Value);
            }
            Console.WriteLine();

            Console.WriteLine("cliext::set contents:");
            ICollection<double> iCollDouble = theClass.GetSetCs();
            foreach (double d in iCollDouble)
            {
                Console.WriteLine(d);
            }
            Console.WriteLine();

            Console.WriteLine("cliext::vector contents:");
            ICollection<int> iCollInt = theClass.GetVectorCs();
            foreach (int i in iCollInt)
            {
                Console.WriteLine(i);
            }
            Console.WriteLine();

            return 0;
        }
    }
}

```

## Output

```
cliext::deque contents:  
a  
b  
  
cliext::list contents:  
3.14159  
2.71828  
  
cliext::map contents:  
0 Hello  
1 World  
  
cliext::set contents:  
2.71828  
3.14159  
  
cliext::vector contents:  
10  
20
```

## See also

[STL/CLR Library Reference](#)

# for each , in

9/20/2022 • 2 minutes to read • [Edit Online](#)

Iterates through an array or collection. This non-standard keyword is available in both C++/CLI and native C++ projects. However, its use isn't recommended. Consider using a standard [Range-based for Statement \(C++\)](#) instead.

## All Runtimes

### Syntax

```
for each ( type identifier in expression ) {  
    statements  
}
```

### Parameters

`type`

The type of `identifier`.

`identifier`

The iteration variable that represents the collection element. When `identifier` is a [Tracking Reference Operator](#), you can modify the element.

`expression`

An array expression or collection. The collection element must be such that the compiler can convert it to the `identifier` type.

`statements`

One or more statements to be executed.

### Remarks

The `for each` statement is used to iterate through a collection. You can modify elements in a collection, but you can't add or delete elements.

The `statements` are executed for each element in the array or collection. After the iteration has been completed for all the elements in the collection, control is transferred to the statement that follows the `for each` block.

`for each` and `in` are [context-sensitive keywords](#).

## Windows Runtime

### Requirements

Compiler option: `/ZW`

### Example

This example shows how to use `for each` to iterate through a string.

```

// for_each_string1.cpp
// compile with: /ZW
#include <stdio.h>
using namespace Platform;

ref struct MyClass {
    property String^ MyStringProperty;
};

int main() {
    String^ MyString = ref new String("abcd");

    for each ( char c in MyString )
        wprintf("%c", c);

    wprintf("/n");

    MyClass^ x = ref new MyClass();
    x->MyStringProperty = "Testing";

    for each( char c in x->MyStringProperty )
        wprintf("%c", c);
}

```

```

abcd

Testing

```

## Common Language Runtime

### Remarks

The CLR syntax is the same as the [All Runtimes](#) syntax, except as follows.

`expression`

A managed array expression or collection. The collection element must be such that the compiler can convert it from [Object](#) to the `identifier` type.

`expression` evaluates to a type that implements [IEnumerable](#), [IEnumerable<T>](#), or a type that defines a `GetEnumerator` method that either returns a type that implements [IEnumerator](#) or declares all of the methods that are defined in [IEnumerator](#).

### Requirements

Compiler option: `/clr`

### Example

This example shows how to use `for each` to iterate through a string.

```
// for_each_string2.cpp
// compile with: /clr
using namespace System;

ref struct MyClass {
    property String ^ MyStringProperty;
};

int main() {
    String ^ MyString = gcnew String("abcd");

    for each ( Char c in MyString )
        Console::Write(c);

    Console::WriteLine();

    MyClass ^ x = gcnew MyClass();
    x->MyStringProperty = "Testing";

    for each( Char c in x->MyStringProperty )
        Console::Write(c);
}
```

```
abcd
```

```
Testing
```

## See also

[Component Extensions for Runtime Platforms](#)

[Range-based for statement \(C++\)](#)

# adapter (STL/CLR)

9/20/2022 • 12 minutes to read • [Edit Online](#)

The STL/CLR header `<cliext/adapter>` specifies two template classes (`collection_adapter` and `range_adapter`), and the template function `make_collection`.

## Syntax

```
#include <cliext/adapter>
```

## Requirements

**Header:** `<cliext/adapter>`

**Namespace:** `cliext`

## Declarations

CLASS	DESCRIPTION
<a href="#">collection_adapter (STL/CLR)</a>	Wraps the Base Class Library (BCL) collection as a range.
<a href="#">range_adapter (STL/CLR)</a>	Wraps the range as a BCL collection.
FUNCTION	DESCRIPTION
<a href="#">make_collection (STL/CLR)</a>	Creates a range adapter using an iterator pair.

## Members

### [collection\\_adapter \(STL/CLR\)](#)

Wraps a .NET collection for use as an STL/CLR container. A `collection_adapter` is a template class that describes a simple STL/CLR container object. It wraps a Base Class Library (BCL) interface, and returns an iterator pair that you use to manipulate the controlled sequence.

#### Syntax

```

template<typename Coll>
    ref class collection_adapter;

template<>
    ref class collection_adapter<
        System::Collections::ICollection>;
template<>
    ref class collection_adapter<
        System::Collections::IEnumerable>;
template<>
    ref class collection_adapter<
        System::Collections::IList>;
template<>
    ref class collection_adapter<
        System::Collections::IDictionary>;
template<typename Value>
    ref class collection_adapter<
        System::Collections::Generic::ICollection<Value>>;
template<typename Value>
    ref class collection_adapter<
        System::Collections::Generic::IEnumerable<Value>>;
template<typename Value>
    ref class collection_adapter<
        System::Collections::Generic::IList<Value>>;
template<typename Key,
         typename Value>
    ref class collection_adapter<
        System::Collections::Generic::IDictionary<Key, Value>>;

```

## Parameters

### *Coll*

The type of the wrapped collection.

## Specializations

SPECIALIZATION	DESCRIPTION
IEnumerable	Sequences through elements.
ICollection	Maintains a group of elements.
IList	Maintains an ordered group of elements.
IDictionary	Maintain a set of {key, value} pairs.
IEnumerable<Value>	Sequences through typed elements.
ICollection<Value>	Maintains a group of typed elements.
IList<Value>	Maintains an ordered group of typed elements.
IDictionary<Value>	Maintains a set of typed {key, value} pairs.

## Members

TYPE DEFINITION	DESCRIPTION
collection_adapter::difference_type (STL/CLR)	The type of a signed distance between two elements.

TYPE DEFINITION	DESCRIPTION
<code>collection_adapter::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>collection_adapter::key_type (STL/CLR)</code>	The type of a dictionary key.
<code>collection_adapter::mapped_type (STL/CLR)</code>	The type of a dictionary value.
<code>collection_adapter::reference (STL/CLR)</code>	The type of a reference to an element.
<code>collection_adapter::size_type (STL/CLR)</code>	The type of a signed distance between two elements.
<code>collection_adapter::value_type (STL/CLR)</code>	The type of an element.
MEMBER FUNCTION	DESCRIPTION
<code>collection_adapter::base (STL/CLR)</code>	Designates the wrapped BCL interface.
<code>collection_adapter::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>collection_adapter::collection_adapter (STL/CLR)</code>	Constructs an adapter object.
<code>collection_adapter::end (STL/CLR)</code>	Designates the end of the controlled sequence.
<code>collection_adapter::size (STL/CLR)</code>	Counts the number of elements.
<code>collection_adapter::swap (STL/CLR)</code>	Swaps the contents of two containers.
OPERATOR	DESCRIPTION
<code>collection_adapter::operator= (STL/CLR)</code>	Replaces the stored BCL handle.

## Remarks

You use this template class to manipulate a BCL container as a STL/CLR container. The `collection_adapter` stores a handle to a BCL interface, which in turn controls a sequence of elements. A `collection_adapter` object `x` returns a pair of input iterators `x.begin()` and `x.end()` that you use to visit the elements, in order. Some of the specializations also let you write `x.size()` to determine the length of the controlled sequence.

## `collection_adapter::base (STL/CLR)`

Designates the wrapped BCL interface.

### Syntax

```
Coll^ base();
```

## Remarks

The member function returns the stored BCL interface handle.

### Example

```

// cliext_collection_adapter_base.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d6x(6, L'x');
    Mycoll c1(%d6x);

    // display initial contents "x x x x x x"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("base() same = {0}", c1.base() == %c1);
    return (0);
}

```

```

x x x x x x
base() same = True

```

## collection\_adapter::begin (STL/CLR)

Designates the beginning of the controlled sequence.

### Syntax

```

iterator begin();

```

### Remarks

The member function returns an input iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence.

### Example

```

// cliext_collection_adapter_begin.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Mycoll::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);
    return (0);
}

```

```

a b c
*begin() = a
*++begin() = b

```

## collection\_adapter::collection\_adapter (STL/CLR)

Constructs an adapter object.

### Syntax

```

collection_adapter();
collection_adapter(collection_adapter<Coll>% right);
collection_adapter(collection_adapter<Coll>^ right);
collection_adapter(Coll^ collection);

```

### Parameters

*collection*

BCL handle to wrap.

*right*

Object to copy.

### Remarks

The constructor:

```
collection_adapter();
```

initializes the stored handle with `nullptr`.

The constructor:

```
collection_adapter(collection_adapter<Coll>% right);
```

initializes the stored handle with `right`. `collection_adapter::base (STL/CLR) ()`.

The constructor:

```
collection_adapter(collection_adapter<Coll>^ right);
```

initializes the stored handle with `right->collection_adapter::base (STL/CLR) ()`.

The constructor:

```
collection_adapter(Coll^ collection);
```

initializes the stored handle with `collection`.

## Example

```
// cliext_collection_adapter_construct.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d6x(6, L'x');

    // construct an empty container
    Mycoll c1;
    System::Console::WriteLine("base() null = {0}", c1.base() == nullptr);

    // construct with a handle
    Mycoll c2(%d6x);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying another container
    Mycoll c3(c2);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying a container handle
    Mycoll c4(%c3);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}
```

```
base() null = True
x x x x x x
x x x x x x
x x x x x x
```

## collection\_adapter::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

## Remarks

The type describes a signed element count.

## Example

```
// cliext_collection_adapter_difference_type.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mycoll::difference_type diff = 0;
    Mycoll::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
```

## collection\_adapter::end (STL/CLR)

Designates the end of the controlled sequence.

## Syntax

```
iterator end();
```

## Remarks

The member function returns an input iterator that points just beyond the end of the controlled sequence.

## Example

```

// cliext_collection_adapter_end.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    Mycoll::iterator it = c1.begin();
    for ( ; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## collection\_adapter::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

### Remarks

The type describes an object of unspecified type `T1` that can serve as an input iterator for the controlled sequence.

### Example

```

// cliext_collection_adapter_iterator.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    Mycoll::iterator it = c1.begin();
    for ( ; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## collection\_adapter::key\_type (STL/CLR)

The type of a dictionary key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter `Key`, in a specialization for `IDictionary` or `IDictionary<Value>`; otherwise it is not defined.

### Example

```

// cliext_collection_adapter_key_type.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
typedef cliext::collection_adapter<
    System::Collections::Generic::IDictionary<wchar_t, int>> Mycoll;
typedef System::Collections::Generic::KeyValuePair<wchar_t,int> Mypair;
int main()
{
    Mymap d1;
    d1.insert(Mymap::make_value(L'a', 1));
    d1.insert(Mymap::make_value(L'b', 2));
    d1.insert(Mymap::make_value(L'c', 3));
    Mycoll c1(%d1);

    // display contents "[a 1] [b 2] [c 3]"
    for each (Mypair elem in c1)
    {
        Mycoll::key_type key = elem.Key;
        Mycoll::mapped_type value = elem.Value;
        System::Console::Write("[{0} {1}] ", key, value);
    }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## collection\_adapter::mapped\_type (STL/CLR)

The type of a dictionary value.

### Syntax

```
typedef Value mapped_type;
```

### Remarks

The type is a synonym for the template parameter `Value`, in a specialization for `IDictionary` or `IDictionary<Value>`; otherwise it is not defined.

### Example

```

// cliext_collection_adapter_mapped_type.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
typedef cliext::collection_adapter<
    System::Collections::Generic::IDictionary<wchar_t, int>> Mycoll;
typedef System::Collections::Generic::KeyValuePair<wchar_t,int> Mypair;
int main()
{
    Mymap d1;
    d1.insert(Mymap::make_value(L'a', 1));
    d1.insert(Mymap::make_value(L'b', 2));
    d1.insert(Mymap::make_value(L'c', 3));
    Mycoll c1(%d1);

    // display contents "[a 1] [b 2] [c 3]"
    for each (Mypair elem in c1)
    {
        Mycoll::key_type key = elem.Key;
        Mycoll::mapped_type value = elem.Value;
        System::Console::Write("[{0} {1}] ", key, value);
    }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## collection\_adapter::operator= (STL/CLR)

Replaces the stored BCL handle.

### Syntax

```
collection_adapter<Coll>% operator=(collection_adapter<Coll>% right);
```

### Parameters

*right*

Adapter to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the stored BCL handle with a copy of the stored BCL handle in *right*.

### Example

```

// cliext_collection_adapter_operator_as.cpp
// compile with: /clr
#include <cliext/adapters>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mycoll c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

## collection\_adapter::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```

typedef value_type% reference;

```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_collection_adapter_reference.cpp
// compile with: /clr
#include <cliext/adapters>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    Mycoll::iterator it = c1.begin();
    for ( ; it != c1.end(); ++it)
    {   // get a reference to an element
        Mycoll::reference ref = *it;
        System::Console::Write("{0} ", ref);
    }
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

## collection\_adapter::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. It is not defined in a specialization for `IEnumerable` or `IEnumerable<Value>`.

### Example

```

// cliext_collection_adapter_size.cpp
// compile with: /clr
#include <cliext/adapters>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d6x(6, L'x');
    Mycoll c1(%d6x);

    // display initial contents "x x x x x x"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

x x x x x x
size() = 6

```

## collection\_adapter::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```

typedef int size_type;

```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_collection_adapter_size_type.cpp
// compile with: /clr
#include <cliext/adapters>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d6x(6, L'x');
    Mycoll c1(%d6x);

    // display initial contents "x x x x x x"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    Mycoll::size_type size = c1.size();
    System::Console::WriteLine("size() = {0}", size);
    return (0);
}

```

```
x x x x x x
size() = 6
```

## collection\_adapter::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```
void swap(collection_adapter<Coll>% right);
```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the stored BCL handles between `*this` and *right*.

### Example

```
// cliext_collection_adapter_swap.cpp
// compile with: /clr
#include <cliext/adAPTER>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::deque<wchar_t> d2(5, L'x');
    Mycoll c2(%d2);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x x x x x
x x x x x
a b c
```

## collection\_adapter::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef Value value_type;
```

### Remarks

The type is a synonym for the template parameter *Value*, if present in the specialization; otherwise it is a synonym for `System::Object^`.

### Example

```
// cliext_collection_adapter_value_type.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display contents "a b c" using value_type
    for (Mycoll::iterator it = c1.begin();
        it != c1.end(); ++it)
    {   // store element in value_type object
        Mycoll::value_type val = *it;

        System::Console::Write("{0} ", val);
    }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## make\_collection (STL/CLR)

Make a `range_adapter` from an iterator pair.

### Syntax

```
template<typename Iter>
range_adapter<Iter> make_collection(Iter first, Iter last);
```

## Parameters

*Iter*

The type of the wrapped iterators.

*first*

First iterator to wrap.

*last*

Second iterator to wrap.

## Remarks

The template function returns `gcnew range_adapter<Iter>(first, last)`. You use it to construct a `range_adapter<Iter>` object from a pair of iterators.

## Example

```
// cliext_make_collection.cpp
// compile with: /clr
#include <cliext/adapters>
#include <cliext/deque>

typedef cliext::deque<wchar_t> Mycont;
typedef cliext::range_adapter<Mycont::iterator> Myrange;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in d1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Collections::ICollection^ p1 =
        cliext::make_collection(d1.begin(), d1.end());
    System::Console::WriteLine("Count = {0}", p1->Count);
    System::Console::WriteLine("IsSynchronized = {0}",
        p1->IsSynchronized);
    System::Console::WriteLine("SyncRoot not nullptr = {0}",
        p1->SyncRoot != nullptr);

    // copy the sequence
    cli::array<System::Object^>^ a1 = gcnew cli::array<System::Object^>(5);

    a1[0] = L'|';
    p1->CopyTo(a1, 1);
    a1[4] = L'|';
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}
```

```
a b c
Count = 3
IsSynchronized = False
SyncRoot not nullptr = True
| a b c |
```

## range\_adapter (STL/CLR)

A template class that wraps a pair of iterators that are used to implement several Base Class Library (BCL) interfaces. You use the range\_adapter to manipulate an STL/CLR range as if it were a BCL collection.

### Syntax

```
template<typename Iter>
ref class range_adapter
: public
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<Value>,
System::Collections::Generic::ICollection<Value>
{ ..... };
```

### Parameters

#### *Iter*

The type associated with the wrapped iterators.

### Members

MEMBER FUNCTION	DESCRIPTION
<a href="#">range_adapter::range_adapter (STL/CLR)</a>	Constructs an adapter object.
OPERATOR	DESCRIPTION
<a href="#">range_adapter::operator= (STL/CLR)</a>	Replaces the stored iterator pair.

### Interfaces

INTERFACE	DESCRIPTION
<a href="#">IEnumerable</a>	Iterates through elements in the collection.
<a href="#">ICollection</a>	Maintains a group of elements.
<a href="#">IEnumerable&lt;T&gt;</a>	Iterates through typed elements in the collection..
<a href="#">ICollection&lt;T&gt;</a>	Maintains a group of typed elements.

### Remarks

The range\_adapter stores a pair of iterators, which in turn delimit a sequence of elements. The object implements four BCL interfaces that let you iterate through the elements, in order. You use this template class to manipulate STL/CLR ranges much like BCL containers.

## range\_adapter::operator= (STL/CLR)

Replaces the stored iterator pair.

### Syntax

```
range_adapter<Iter>% operator=(range_adapter<Iter>% right);
```

### Parameters

*right*

Adapter to copy.

## Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the stored iterator pair with a copy of the stored iterator pair in *right*.

## Example

```
// cliext_range_adapter_operator_as.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::deque<wchar_t> Mycont;
typedef cliext::range_adapter<Mycont::iterator> Myrange;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Myrange c1(d1.begin(), d1.end());

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myrange c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

## range\_adapter::range\_adapter (STL/CLR)

Constructs an adapter object.

### Syntax

```
range_adapter();
range_adapter(range_adapter<Iter>% right);
range_adapter(range_adapter<Iter>^ right);
range_adapter(Iter first, Iter last);
```

### Parameters

*first*

First iterator to wrap.

*last*

Second iterator to wrap.

*right*

Object to copy.

## Remarks

The constructor:

```
range_adapter();
```

initializes the stored iterator pair with default constructed iterators.

The constructor:

```
range_adapter(range_adapter<Iter>% right);
```

initializes the stored iterator pair by copying the pair stored in *right*.

The constructor:

```
range_adapter(range_adapter<Iter>^ right);
```

initializes the stored iterator pair by copying the pair stored in `*right`.

The constructor:

```
range_adapter(Iter^ first, last);
```

initializes the stored iterator pair with *first* and *last*.

## Example

```
// cliext_range_adapter_construct.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::deque<wchar_t> Mycont;
typedef cliext::range_adapter<Mycont::iterator> Myrange;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');

    // construct an empty adapter
    Myrange c1;

    // construct with an iterator pair
    Myrange c2(d1.begin(), d1.end());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying another adapter
    Myrange c3(c2);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying an adapter handle
    Myrange c4(%c3);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}
```

```
a b c
a b c
a b c
```

# algorithm (STL/CLR)

9/20/2022 • 29 minutes to read • [Edit Online](#)

Defines STL/CLR container template functions that perform algorithms.

## Syntax

```
#include <cliext/algorithm>
```

## Requirements

**Header:** <cliext/algorithm>

**Namespace:** cliext

## Declarations

FUNCTION	DESCRIPTION
<a href="#">adjacent_find (STL/CLR)</a>	Searches for two adjacent elements that are equal.
<a href="#">binary_search (STL/CLR)</a>	Tests whether a sorted sequence contains a given value.
<a href="#">copy (STL/CLR)</a>	Copies values from a source range to a destination range, iterating in the forward direction.
<a href="#">copy_backward (STL/CLR)</a>	Copies values from a source range to a destination range, iterating in the backward direction.
<a href="#">count (STL/CLR)</a>	Returns the number of elements in a range whose values match a specified value.
<a href="#">count_if (STL/CLR)</a>	Returns the number of elements in a range whose values match a specified condition.
<a href="#">equal (STL/CLR)</a>	Compares two ranges, element by element.
<a href="#">equal_range (STL/CLR)</a>	Searches an ordered sequence of values and returns two positions that delimit a subsequence of values that are all equal to a given element.
<a href="#">fill (STL/CLR)</a>	Assigns the same new value to every element in a specified range.
<a href="#">fill_n (STL/CLR)</a>	Assigns a new value to a specified number of elements in a range beginning with a particular element.
<a href="#">find (STL/CLR)</a>	Returns the position of the first occurrence of a specified value.

FUNCTION	DESCRIPTION
<a href="#">find_end (STL/CLR)</a>	Returns the last subsequence in a range that is identical to a specified sequence.
<a href="#">find_first_of (STL/CLR)</a>	Searches a range for the first occurrence of any one of a given range of elements.
<a href="#">find_if (STL/CLR)</a>	Returns the position of the first element in a sequence of values where the element satisfies a specified condition.
<a href="#">for_each (STL/CLR)</a>	Applies a specified function object to each element in a sequence of values and returns the function object.
<a href="#">generate (STL/CLR)</a>	Assigns the values generated by a function object to each element in a sequence of values.
<a href="#">generate_n (STL/CLR)</a>	Assigns the values generated by a function object to a specified number of elements.
<a href="#">includes (STL/CLR)</a>	Tests whether one sorted range contains all the elements in a second sorted range.
<a href="#">inplace_merge (STL/CLR)</a>	Combines the elements from two consecutive sorted ranges into a single sorted range.
<a href="#">iter_swap (STL/CLR)</a>	Exchanges two values referred to by a pair of specified iterators.
<a href="#">lexicographical_compare (STL/CLR)</a>	Compares two sequences, element by element, identifying which sequence is the lesser of the two.
<a href="#">lower_bound (STL/CLR)</a>	Finds the position of the first element in an ordered sequence of values that has a value greater than or equal to a specified value.
<a href="#">make_heap (STL/CLR)</a>	Converts elements from a specified range into a heap where the first element on the heap is the largest.
<a href="#">max (STL/CLR)</a>	Compares two objects and returns the greater of the two.
<a href="#">max_element (STL/CLR)</a>	Finds the largest element in a specified sequence of values.
<a href="#">merge (STL/CLR)</a>	Combines all the elements from two sorted source ranges into a single, sorted destination range.
<a href="#">min (STL/CLR)</a>	Compares two objects and returns the lesser of the two.
<a href="#">min_element (STL/CLR)</a>	Finds the smallest element in a specified sequence of values.
<a href="#">mismatch (STL/CLR)</a>	Compares two ranges element by element and returns the first position where a difference occurs.

FUNCTION	DESCRIPTION
<a href="#">next_permutation (STL/CLR)</a>	Reorders the elements in a range so that the original ordering is replaced by the lexicographically next greater permutation if it exists.
<a href="#">nth_element (STL/CLR)</a>	Partitions a sequence of elements, correctly locating the $n$ th element of the sequence so that all the elements in front of it are less than or equal to it and all the elements that follow it are greater than or equal to it.
<a href="#">partial_sort (STL/CLR)</a>	Arranges a specified number of the smaller elements in a range into nondescending order.
<a href="#">partial_sort_copy (STL/CLR)</a>	Copies elements from a source range into a destination range such that the elements from the source range are ordered.
<a href="#">partition (STL/CLR)</a>	Arranges elements in a range such that those elements that satisfy a unary predicate precede those that fail to satisfy it.
<a href="#">pop_heap (STL/CLR)</a>	Moves the largest element from the front of a heap to the end and then forms a new heap from the remaining elements.
<a href="#">prev_permutation (STL/CLR)</a>	Reorders a sequence of elements so that the original ordering is replaced by the lexicographically previous greater permutation if it exists.
<a href="#">push_heap (STL/CLR)</a>	Adds an element that is at the end of a range to an existing heap consisting of the prior elements in the range.
<a href="#">random_shuffle (STL/CLR)</a>	Rearranges a sequence of $N$ elements in a range into one of $N!$ possible arrangements selected at random.
<a href="#">remove (STL/CLR)</a>	Deletes a specified value from a given range without disturbing the order of the remaining elements and returns the end of a new range free of the specified value.
<a href="#">remove_copy (STL/CLR)</a>	Copies elements from a source range to a destination range, except that elements of a specified value are not copied, without disturbing the order of the remaining elements.
<a href="#">remove_copy_if (STL/CLR)</a>	Copies elements from a source range to a destination range, except those that satisfy a predicate, without disturbing the order of the remaining elements.
<a href="#">remove_if (STL/CLR)</a>	Deletes elements that satisfy a predicate from a given range without disturbing the order of the remaining elements.
<a href="#">replace (STL/CLR)</a>	Replaces elements in a range that match a specified value with a new value.
<a href="#">replace_copy (STL/CLR)</a>	Copies elements from a source range to a destination range, replacing elements that match a specified value with a new value.

FUNCTION	DESCRIPTION
<a href="#">replace_copy_if (STL/CLR)</a>	Examines each element in a source range and replaces it if it satisfies a specified predicate while copying the result into a new destination range.
<a href="#">replace_if (STL/CLR)</a>	Examines each element in a range and replaces it if it satisfies a specified predicate.
<a href="#">reverse (STL/CLR)</a>	Reverses the order of the elements within a range.
<a href="#">reverse_copy (STL/CLR)</a>	Reverses the order of the elements within a source range while copying them into a destination range.
<a href="#">rotate (STL/CLR)</a>	Exchanges the elements in two adjacent ranges.
<a href="#">rotate_copy (STL/CLR)</a>	Exchanges the elements in two adjacent ranges within a source range and copies the result to a destination range.
<a href="#">search (STL/CLR)</a>	Searches for the first occurrence of a sequence within a target range whose elements are equal to those in a given sequence of elements or whose elements are equivalent in a sense specified by a binary predicate to the elements in the given sequence.
<a href="#">search_n (STL/CLR)</a>	Searches for the first subsequence in a range that of a specified number of elements having a particular value or a relation to that value as specified by a binary predicate.
<a href="#">set_difference (STL/CLR)</a>	Unites all of the elements that belong to one sorted source range, but not to a second sorted source range, into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.
<a href="#">set_intersection (STL/CLR)</a>	Unites all of the elements that belong to both sorted source ranges into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.
<a href="#">set_symmetric_difference (STL/CLR)</a>	Unites all of the elements that belong to one, but not both, of the sorted source ranges into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.
<a href="#">set_union (STL/CLR)</a>	Unites all of the elements that belong to at least one of two sorted source ranges into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.
<a href="#">sort (STL/CLR)</a>	Arranges the elements in a specified range into a nondescending order or according to an ordering criterion specified by a binary predicate.
<a href="#">sort_heap (STL/CLR)</a>	Converts a heap into a sorted range.

FUNCTION	DESCRIPTION
<a href="#">stable_partition (STL/CLR)</a>	Classifies elements in a range into two disjoint sets, with those elements satisfying a unary predicate preceding those that fail to satisfy it, preserving the relative order of equivalent elements.
<a href="#">stable_sort (STL/CLR)</a>	Arranges the elements in a specified range into a nondescending order or according to an ordering criterion specified by a binary predicate and preserves the relative ordering of equivalent elements.
<a href="#">swap (STL/CLR)</a>	Exchanges the values of the elements between two types of objects, assigning the contents of the first object to the second object and the contents of the second to the first.
<a href="#">swap_ranges (STL/CLR)</a>	Exchanges the elements of one range with the elements of another, equal sized range.
<a href="#">transform (STL/CLR)</a>	Applies a specified function object to each element in a source range or to a pair of elements from two source ranges and copies the return values of the function object into a destination range.
<a href="#">unique (STL/CLR)</a>	Removes duplicate elements that are adjacent to each other in a specified range.
<a href="#">unique_copy (STL/CLR)</a>	Copies elements from a source range into a destination range except for the duplicate elements that are adjacent to each other.
<a href="#">upper_bound (STL/CLR)</a>	Finds the position of the first element in an ordered range that has a value that is greater than a specified value, where the ordering criterion may be specified by a binary predicate.

## Members

### adjacent\_find (STL/CLR)

Searches for two adjacent elements that are either equal or satisfy a specified condition.

#### Syntax

```
template<class _FwdIt> inline
    _FwdIt adjacent_find(_FwdIt _First, _FwdIt _Last);
template<class _FwdIt, class _Pr> inline
    _FwdIt adjacent_find(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

#### Remarks

This function behaves the same as the C++ Standard Library function [adjacent\\_find](#). For more information, see [adjacent\\_find](#).

### binary\_search (STL/CLR)

Tests whether there is an element in a sorted range that is equal to a specified value or that is equivalent to it in a sense specified by a binary predicate.

## Syntax

```
template<class _FwdIt, class _Ty> inline
    bool binary_search(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);
template<class _FwdIt, class _Ty, class _Pr> inline
    bool binary_search(_FwdIt _First, _FwdIt _Last,
        const _Ty% _Val, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [binary\\_search](#). For more information, see [binary\\_search](#).

## copy (STL/CLR)

Assigns the values of elements from a source range to a destination range, iterating through the source sequence of elements and assigning them new positions in a forward direction.

## Syntax

```
template<class _InIt, class _OutIt> inline
    _OutIt copy(_InIt _First, _InIt _Last, _OutIt _Dest);
```

## Remarks

This function behaves the same as the C++ Standard Library function [copy](#). For more information, see [copy](#).

## copy\_backward (STL/CLR)

Assigns the values of elements from a source range to a destination range, iterating through the source sequence of elements and assigning them new positions in a backward direction.

## Syntax

```
template<class _BidIt1, class _BidIt2> inline
    _BidIt2 copy_backward(_BidIt1 _First, _BidIt1 _Last,
        _BidIt2 _Dest);
```

## Remarks

This function behaves the same as the C++ Standard Library function [copy\\_backward](#). For more information, see [copy\\_backward](#).

## count (STL/CLR)

Returns the number of elements in a range whose values match a specified value.

## Syntax

```
template<class _InIt, class _Ty> inline
    typename iterator_traits<_InIt>::difference_type
        count(_InIt _First, _InIt _Last, const _Ty% _Val);
```

## Remarks

This function behaves the same as the C++ Standard Library function [count](#). For more information, see [count](#).

## count\_if (STL/CLR)

Returns the number of elements in a range whose values match a specified condition.

## Syntax

```
template<class _InIt, class _Pr> inline
    typename iterator_traits<_InIt>::difference_type
        count_if(_InIt _First, _InIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `count_if`. For more information, see [count\\_if](#).

## equal (STL/CLR)

Compares two ranges element by element either for equality or equivalence in a sense specified by a binary predicate.

## Syntax

```
template<class _InIt1, class _InIt2> inline
    bool equal(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2);
template<class _InIt1, class _InIt2, class _Pr> inline
    bool equal(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
               _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `equal`. For more information, see [equal](#).

## equal\_range (STL/CLR)

Finds a pair of positions in an ordered range, the first less than or equivalent to the position of a specified element and the second greater than the element's position, where the sense of equivalence or ordering used to establish the positions in the sequence may be specified by a binary predicate.

## Syntax

```
template<class _FwdIt, class _Ty> inline
    _PAIR_TYPE(_FwdIt) equal_range(_FwdIt _First, _FwdIt _Last,
                                   const _Ty% _Val);
template<class _FwdIt, class _Ty, class _Pr> inline
    _PAIR_TYPE(_FwdIt) equal_range(_FwdIt _First, _FwdIt _Last,
                                   const _Ty% _Val, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `equal_range`. For more information, see [equal\\_range](#).

## fill (STL/CLR)

Assigns the same new value to every element in a specified range.

## Syntax

```
template<class _FwdIt, class _Ty> inline
    void fill(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);
```

## Remarks

This function behaves the same as the C++ Standard Library function [fill](#). For more information, see [fill](#).

## fill\_n (STL/CLR)

Assigns a new value to a specified number of elements in a range beginning with a particular element.

### Syntax

```
template<class _OutIt, class _Diff, class _Ty> inline
void fill_n(_OutIt _First, _Diff _Count, const _Ty% _Val);
```

## Remarks

This function behaves the same as the C++ Standard Library function [fill\\_n](#). For more information, see [fill\\_n](#).

## find (STL/CLR)

Locates the position of the first occurrence of an element in a range that has a specified value.

### Syntax

```
template<class _InIt, class _Ty> inline
_InIt find(_InIt _First, _InIt _Last, const _Ty% _Val);
```

## Remarks

This function behaves the same as the C++ Standard Library function [find](#). For more information, see [find](#).

## find\_end (STL/CLR)

Looks in a range for the last subsequence that is identical to a specified sequence or that is equivalent in a sense specified by a binary predicate.

### Syntax

```
template<class _FwdIt1, class _FwdIt2> inline
_FwdIt1 find_end(_FwdIt1 _First1, _FwdIt1 _Last1,
                  _FwdIt2 _First2, _FwdIt2 _Last2);
template<class _FwdIt1, class _FwdIt2, class _Pr> inline
_FwdIt1 find_end(_FwdIt1 _First1, _FwdIt1 _Last1,
                  _FwdIt2 _First2, _FwdIt2 _Last2, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [find\\_end](#). For more information, see [find\\_end](#).

## find\_first\_of (STL/CLR)

Searches for the first occurrence of any of several values within a target range or for the first occurrence of any of several elements that are equivalent in a sense specified by a binary predicate to a specified set of the elements.

### Syntax

```
template<class _FwdIt1, class _FwdIt2> inline
    _FwdIt1 find_first_of(_FwdIt1 _First1, _FwdIt1 _Last1,
    _FwdIt2 _First2, _FwdIt2 _Last2);
template<class _FwdIt1, class _FwdIt2, class _Pr> inline
    _FwdIt1 find_first_of(_FwdIt1 _First1, _FwdIt1 _Last1,
    _FwdIt2 _First2, _FwdIt2 _Last2, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [find\\_first\\_of](#). For more information, see [find\\_first\\_of](#).

## find\_if (STL/CLR)

Locates the position of the first occurrence of an element in a range that satisfies a specified condition.

### Syntax

```
template<class _InIt, class _Pr> inline
    _InIt find_if(_InIt _First, _InIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [find\\_if](#). For more information, see [find\\_if](#).

## for\_each (STL/CLR)

Applies a specified function object to each element in a forward order within a range and returns the function object.

### Syntax

```
template<class _InIt, class _Fn1> inline
    _Fn1 for_each(_InIt _First, _InIt _Last, _Fn1 _Func);
```

## Remarks

This function behaves the same as the C++ Standard Library function [for\\_each](#). For more information, see [for\\_each](#).

## generate (STL/CLR)

Assigns the values generated by a function object to each element in a range.

### Syntax

```
template<class _FwdIt, class _Fn0> inline
    void generate(_FwdIt _First, _FwdIt _Last, _Fn0 _Func);
```

## Remarks

This function behaves the same as the C++ Standard Library function [generate](#). For more information, see [generate](#).

## generate\_n (STL/CLR)

Assigns the values generated by a function object to a specified number of elements in a range and returns to the position one past the last assigned value.

### Syntax

```
template<class _OutIt, class _Diff, class _Fn0> inline
void generate_n(_OutIt _Dest, _Diff _Count, _Fn0 _Func);
```

### Remarks

This function behaves the same as the C++ Standard Library function [generate\\_n](#). For more information, see [generate\\_n](#).

## includes (STL/CLR)

Tests whether one sorted range contains all the elements contained in a second sorted range, where the ordering or equivalence criterion between elements may be specified by a binary predicate.

### Syntax

```
template<class _InIt1, class _InIt2> inline
bool includes(_InIt1 _First1, _InIt1 _Last1,
              _InIt2 _First2, _InIt2 _Last2);
template<class _InIt1, class _InIt2, class _Pr> inline
bool includes(_InIt1 _First1, _InIt1 _Last1,
              _InIt2 _First2, _InIt2 _Last2, _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [includes](#). For more information, see [includes](#).

## inplace\_merge (STL/CLR)

Combines the elements from two consecutive sorted ranges into a single sorted range, where the ordering criterion may be specified by a binary predicate.

### Syntax

```
template<class _BidIt> inline
void inplace_merge(_BidIt _First, _BidIt _Mid, _BidIt _Last);
template<class _BidIt, class _Pr> inline
void inplace_merge(_BidIt _First, _BidIt _Mid, _BidIt _Last,
                  _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [inplace\\_merge](#). For more information, see [inplace\\_merge](#).

## iter\_swap (STL/CLR)

Exchanges two values referred to by a pair of specified iterators.

### Syntax

```
template<class _FwdIt1, class _FwdIt2> inline
void iter_swap(_FwdIt1 _Left, _FwdIt2 _Right);
```

## Remarks

This function behaves the same as the C++ Standard Library function `iter_swap`. For more information, see [iter\\_swap](#).

## lexicographical\_compare (STL/CLR)

Compares element by element between two sequences to determine which is lesser of the two.

### Syntax

```
template<class _InIt1, class _InIt2> inline
    bool lexicographical_compare(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2);
template<class _InIt1, class _InIt2, class _Pr> inline
    bool lexicographical_compare(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `lexicographical_compare`. For more information, see [lexicographical\\_compare](#).

## lower\_bound (STL/CLR)

Finds the position of the first element in an ordered range that has a value less than or equivalent to a specified value, where the ordering criterion may be specified by a binary predicate.

### Syntax

```
template<class _FwdIt, class _Ty> inline
    _FwdIt lower_bound(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);
template<class _FwdIt, class _Ty, class _Pr> inline
    _FwdIt lower_bound(_FwdIt _First, _FwdIt _Last,
        const _Ty% _Val, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `lower_bound`. For more information, see [lower\\_bound](#).

## make\_heap (STL/CLR)

Converts elements from a specified range into a heap in which the first element is the largest and for which a sorting criterion may be specified with a binary predicate.

### Syntax

```
template<class _RanIt> inline
    void make_heap(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void make_heap(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `make_heap`. For more information, see [make\\_heap](#).

## max (STL/CLR)

Compares two objects and returns the larger of the two, where the ordering criterion may be specified by a binary predicate.

## Syntax

```
template<class _Ty> inline
    const _Ty max(const _Ty% _Left, const _Ty% _Right);
template<class _Ty, class _Pr> inline
    const _Ty max(const _Ty% _Left, const _Ty% _Right, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [max](#). For more information, see [max](#).

## max\_element (STL/CLR)

Finds the first occurrence of largest element in a specified range where the ordering criterion may be specified by a binary predicate.

## Syntax

```
template<class _FwdIt> inline
    _FwdIt max_element(_FwdIt _First, _FwdIt _Last);
template<class _FwdIt, class _Pr> inline
    _FwdIt max_element(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [max\\_element](#). For more information, see [max\\_element](#).

## merge (STL/CLR)

Combines all the elements from two sorted source ranges into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.

## Syntax

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt merge(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt merge(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [merge](#). For more information, see [merge](#).

## min (STL/CLR)

Compares two objects and returns the lesser of the two, where the ordering criterion may be specified by a binary predicate.

## Syntax

```

template<class _Ty> inline
    const _Ty min(const _Ty% _Left, const _Ty% _Right);
template<class _Ty, class _Pr> inline
    const _Ty min(const _Ty% _Left, const _Ty% _Right, _Pr _Pred);

```

## Remarks

This function behaves the same as the C++ Standard Library function [min](#). For more information, see [min](#).

## min\_element (STL/CLR)

Finds the first occurrence of smallest element in a specified range where the ordering criterion may be specified by a binary predicate.

### Syntax

```

template<class _FwdIt> inline
    _FwdIt min_element(_FwdIt _First, _FwdIt _Last);
template<class _FwdIt, class _Pr> inline
    _FwdIt min_element(_FwdIt _First, _FwdIt _Last, _Pr _Pred);

```

## Remarks

This function behaves the same as the C++ Standard Library function [min\\_element](#). For more information, see [min\\_element](#).

## mismatch (STL/CLR)

Compares two ranges element by element either for equality or equivalent in a sense specified by a binary predicate and locates the first position where a difference occurs.

### Syntax

```

template<class _InIt1, class _InIt2> inline
    _PAIR_TYPE(_InIt1)
        mismatch(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2);
template<class _InIt1, class _InIt2, class _Pr> inline
    _PAIR_TYPE(_InIt1)
        mismatch(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
            _Pr _Pred);

```

## Remarks

This function behaves the same as the C++ Standard Library function [mismatch](#). For more information, see [mismatch](#).

## next\_permutation (STL/CLR)

Reorders the elements in a range so that the original ordering is replaced by the lexicographically next greater permutation if it exists, where the sense of next may be specified with a binary predicate.

### Syntax

```

template<class _BidIt> inline
    bool next_permutation(_BidIt _First, _BidIt _Last);
template<class _BidIt, class _Pr> inline
    bool next_permutation(_BidIt _First, _BidIt _Last, _Pr _Pred);

```

## Remarks

This function behaves the same as the C++ Standard Library function `next_permutation`. For more information, see [next\\_permutation](#).

## nth\_element (STL/CLR)

Partitions a range of elements, correctly locating the `n`th element of the sequence in the range so that all the elements in front of it are less than or equal to it and all the elements that follow it in the sequence are greater than or equal to it.

### Syntax

```
template<class _RanIt> inline
    void nth_element(_RanIt _First, _RanIt _Nth, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void nth_element(_RanIt _First, _RanIt _Nth, _RanIt _Last,
        _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `nth_element`. For more information, see [nth\\_element](#).

## partial\_sort (STL/CLR)

Arranges a specified number of the smaller elements in a range into a nondescending order or according to an ordering criterion specified by a binary predicate.

### Syntax

```
template<class _RanIt> inline
    void partial_sort(_RanIt _First, _RanIt _Mid, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void partial_sort(_RanIt _First, _RanIt _Mid, _RanIt _Last,
        _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `partial_sort`. For more information, see [partial\\_sort](#).

## partial\_sort\_copy (STL/CLR)

Copies elements from a source range into a destination range where the source elements are ordered by either less than or another specified binary predicate.

### Syntax

```
template<class _InIt, class _RanIt> inline
    _RanIt partial_sort_copy(_InIt _First1, _InIt _Last1,
        _RanIt _First2, _RanIt _Last2);
template<class _InIt, class _RanIt, class _Pr> inline
    _RanIt partial_sort_copy(_InIt _First1, _InIt _Last1,
        _RanIt _First2, _RanIt _Last2, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `partial_sort_copy`. For more information,

see [partial\\_sort\\_copy](#).

## partition (STL/CLR)

Classifies elements in a range into two disjoint sets, with those elements satisfying a unary predicate preceding those that fail to satisfy it.

### Syntax

```
template<class _BidIt, class _Pr> inline  
_BidIt partition(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [partition](#). For more information, see [partition](#).

## pop\_heap (STL/CLR)

Removes the largest element from the front of a heap to the next-to-last position in the range and then forms a new heap from the remaining elements.

### Syntax

```
template<class _RanIt> inline  
void pop_heap(_RanIt _First, _RanIt _Last);  
template<class _RanIt, class _Pr> inline  
void pop_heap(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [pop\\_heap](#). For more information, see [pop\\_heap](#).

## prev\_permutation (STL/CLR)

Reorders the elements in a range so that the original ordering is replaced by the lexicographically next greater permutation if it exists, where the sense of next may be specified with a binary predicate.

### Syntax

```
template<class _BidIt> inline  
bool prev_permutation(_BidIt _First, _BidIt _Last);  
template<class _BidIt, class _Pr> inline  
bool prev_permutation(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [prev\\_permutation](#). For more information, see [prev\\_permutation](#).

## push\_heap (STL/CLR)

Adds an element that is at the end of a range to an existing heap consisting of the prior elements in the range.

### Syntax

```
template<class _RanIt> inline
    void push_heap(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void push_heap(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [push\\_heap](#). For more information, see [push\\_heap](#).

## random\_shuffle (STL/CLR)

Rearranges a sequence of  $N$  elements in a range into one of  $N!$  possible arrangements selected at random.

### Syntax

```
template<class _RanIt> inline
    void random_shuffle(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Fn1> inline
    void random_shuffle(_RanIt _First, _RanIt _Last, _Fn1% _Func);
```

## Remarks

This function behaves the same as the C++ Standard Library function [random\\_shuffle](#). For more information, see [random\\_shuffle](#).

## remove (STL/CLR)

Eliminates a specified value from a given range without disturbing the order of the remaining elements and returning the end of a new range free of the specified value.

### Syntax

```
template<class _FwdIt, class _Ty> inline
    _FwdIt remove(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);
```

## Remarks

This function behaves the same as the C++ Standard Library function [remove](#). For more information, see [remove](#).

## remove\_copy (STL/CLR)

Copies elements from a source range to a destination range, except that elements of a specified value are not copied, without disturbing the order of the remaining elements and returning the end of a new destination range.

### Syntax

```
template<class _InIt, class _OutIt, class _Ty> inline
    _OutIt remove_copy(_InIt _First, _InIt _Last,
        _OutIt _Dest, const _Ty% _Val);
```

## Remarks

This function behaves the same as the C++ Standard Library function [remove\\_copy](#). For more information, see [remove\\_copy](#).

## remove\_copy\_if (STL/CLR)

Copies elements from a source range to a destination range, except that satisfying a predicate are not copied, without disturbing the order of the remaining elements and returning the end of a new destination range.

### Syntax

```
template<class _InIt, class _OutIt, class _Pr> inline
    _OutIt remove_copy_if(_InIt _First, _InIt _Last, _OutIt _Dest,
        _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [remove\\_copy\\_if](#). For more information, see [remove\\_copy\\_if](#).

## remove\_if (STL/CLR)

Eliminates elements that satisfy a predicate from a given range without disturbing the order of the remaining elements and returning the end of a new range free of the specified value.

### Syntax

```
template<class _FwdIt, class _Pr> inline
    _FwdIt remove_if(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [remove\\_if](#). For more information, see [remove\\_if](#).

## replace (STL/CLR)

Examines each element in a range and replaces it if it matches a specified value.

### Syntax

```
template<class _FwdIt, class _Ty> inline
    void replace(_FwdIt _First, _FwdIt _Last,
        const _Ty% _Oldval, const _Ty% _Newval);
```

### Remarks

This function behaves the same as the C++ Standard Library function [replace](#). For more information, see [replace](#).

## replace\_copy (STL/CLR)

Examines each element in a source range and replaces it if it matches a specified value while copying the result into a new destination range.

### Syntax

```
template<class _InIt, class _OutIt, class _Ty> inline
    _OutIt replace_copy(_InIt _First, _InIt _Last, _OutIt _Dest,
        const _Ty% _Oldval, const _Ty% _Newval);
```

## Remarks

This function behaves the same as the C++ Standard Library function `replace_copy`. For more information, see [replace\\_copy](#).

## replace\_copy\_if (STL/CLR)

Examines each element in a source range and replaces it if it satisfies a specified predicate while copying the result into a new destination range.

### Syntax

```
template<class _InIt, class _OutIt, class _Pr, class _Ty> inline  
_OutIt replace_copy_if(_InIt _First, _InIt _Last, _OutIt _Dest,  
_Pr _Pred, const _Ty% _Val);
```

## Remarks

This function behaves the same as the C++ Standard Library function `replace_copy_if`. For more information, see [replace\\_copy\\_if](#).

## replace\_if (STL/CLR)

Examines each element in a range and replaces it if it satisfies a specified predicate.

### Syntax

```
template<class _FwdIt, class _Pr, class _Ty> inline  
void replace_if(_FwdIt _First, _FwdIt _Last, _Pr _Pred,  
const _Ty% _Val);
```

## Remarks

This function behaves the same as the C++ Standard Library function `replace_if`. For more information, see [replace\\_if](#).

## reverse (STL/CLR)

Reverses the order of the elements within a range.

### Syntax

```
template<class _BidIt> inline  
void reverse(_BidIt _First, _BidIt _Last);
```

## Remarks

This function behaves the same as the C++ Standard Library function `reverse`. For more information, see [reverse](#).

## reverse\_copy (STL/CLR)

Reverses the order of the elements within a source range while copying them into a destination range.

### Syntax

```
template<class _BidIt, class _OutIt> inline
    _OutIt reverse_copy(_BidIt _First, _BidIt _Last, _OutIt _Dest);
```

## Remarks

This function behaves the same as the C++ Standard Library function [reverse\\_copy](#). For more information, see [reverse\\_copy](#).

## rotate (STL/CLR)

Exchanges the elements in two adjacent ranges.

### Syntax

```
template<class _FwdIt> inline
    void rotate(_FwdIt _First, _FwdIt _Mid, _FwdIt _Last);
```

## Remarks

This function behaves the same as the C++ Standard Library function [rotate](#). For more information, see [rotate](#).

## rotate\_copy (STL/CLR)

Exchanges the elements in two adjacent ranges within a source range and copies the result to a destination range.

### Syntax

```
template<class _FwdIt, class _OutIt> inline
    _OutIt rotate_copy(_FwdIt _First, _FwdIt _Mid, _FwdIt _Last,
        _OutIt _Dest);
```

## Remarks

This function behaves the same as the C++ Standard Library function [rotate\\_copy](#). For more information, see [rotate\\_copy](#).

## search (STL/CLR)

Searches for the first occurrence of a sequence within a target range whose elements are equal to those in a given sequence of elements or whose elements are equivalent in a sense specified by a binary predicate to the elements in the given sequence.

### Syntax

```
template<class _FwdIt1, class _FwdIt2> inline
    _FwdIt1 search(_FwdIt1 _First1, _FwdIt1 _Last1,
        _FwdIt2 _First2, _FwdIt2 _Last2);
template<class _FwdIt1, class _FwdIt2, class _Pr> inline
    _FwdIt1 search(_FwdIt1 _First1, _FwdIt1 _Last1,
        _FwdIt2 _First2, _FwdIt2 _Last2, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [search](#). For more information, see [search](#).

## search\_n (STL/CLR)

Searches for the first subsequence in a range that of a specified number of elements having a particular value or a relation to that value as specified by a binary predicate.

### Syntax

```
template<class _FwdIt1, class _Diff2, class _Ty> inline
    _FwdIt1 search_n(_FwdIt1 _First1, _FwdIt1 _Last1,
        _Diff2 _Count, const _Ty& _Val);
template<class _FwdIt1, class _Diff2, class _Ty, class _Pr> inline
    _FwdIt1 search_n(_FwdIt1 _First1, _FwdIt1 _Last1,
        _Diff2 _Count, const _Ty& _Val, _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [search\\_n](#). For more information, see [search\\_n](#).

## set\_difference (STL/CLR)

Unites all of the elements that belong to one sorted source range, but not to a second sorted source range, into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.

### Syntax

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt set_difference(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt set_difference(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [set\\_difference](#). For more information, see [set\\_difference](#).

## set\_intersection (STL/CLR)

Unites all of the elements that belong to both sorted source ranges into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.

### Syntax

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt set_intersection(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt set_intersection(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

### Remarks

This function behaves the same as the C++ Standard Library function [set\\_intersection](#). For more information, see [set\\_intersection](#).

## set\_symmetric\_difference (STL/CLR)

Unites all of the elements that belong to one, but not both, of the sorted source ranges into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.

## Syntax

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt set_symmetric_difference(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt set_symmetric_difference(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [set\\_symmetric\\_difference](#). For more information, see [set\\_symmetric\\_difference](#).

## set\_union (STL/CLR)

Unites all of the elements that belong to at least one of two sorted source ranges into a single, sorted destination range, where the ordering criterion may be specified by a binary predicate.

## Syntax

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt set_union(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt set_union(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [set\\_union](#). For more information, see [set\\_union](#).

## sort (STL/CLR)

Arranges the elements in a specified range into a nondescending order or according to an ordering criterion specified by a binary predicate.

## Syntax

```
template<class _RanIt> inline
    void sort(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void sort(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [sort](#). For more information, see [sort](#).

## sort\_heap (STL/CLR)

Converts a heap into a sorted range.

## Syntax

```
template<class _RanIt> inline
    void sort_heap(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void sort_heap(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `sort_heap`. For more information, see [sort\\_heap](#).

## stable\_partition (STL/CLR)

Classifies elements in a range into two disjoint sets, with those elements satisfying a unary predicate preceding those that fail to satisfy it, preserving the relative order of equivalent elements.

## Syntax

```
template<class _BidIt, class _Pr> inline
    _BidIt stable_partition(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `stable_partition`. For more information, see [stable\\_partition](#).

## stable\_sort (STL/CLR)

Arranges the elements in a specified range into a nondescending order or according to an ordering criterion specified by a binary predicate and preserves the relative ordering of equivalent elements.

## Syntax

```
template<class _BidIt> inline
    void stable_sort(_BidIt _First, _BidIt _Last);
template<class _BidIt, class _Pr> inline
    void stable_sort(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function `stable_sort`. For more information, see [stable\\_sort](#).

## swap (STL/CLR)

Exchanges the values of the elements between two types of objects, assigning the contents of the first object to the second object and the contents of the second to the first.

## Syntax

```
<class _Ty> inline
    void swap(_Ty% _Left, _Ty% _Right);
```

## Remarks

This function behaves the same as the C++ Standard Library function `swap`. For more information, see [swap](#).

## swap\_ranges (STL/CLR)

Exchanges the elements of one range with the elements of another, equal sized range.

## Syntax

```
template<class _FwdIt1, class _FwdIt2> inline
    _FwdIt2 swap_ranges(_FwdIt1 _First1, _FwdIt1 _Last1,
        _FwdIt2 _First2);
```

## Remarks

This function behaves the same as the C++ Standard Library function [swap\\_ranges](#). For more information, see [swap\\_ranges](#).

## transform (STL/CLR)

Applies a specified function object to each element in a source range or to a pair of elements from two source ranges and copies the return values of the function object into a destination range.

## Syntax

```
template<class _InIt, class _OutIt, class _Fn1> inline
    _OutIt transform(_InIt _First, _InIt _Last, _OutIt _Dest,
        _Fn1 _Func);
template<class _InIt1, class _InIt2, class _OutIt, class _Fn2> inline
    _OutIt transform(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
        _OutIt _Dest, _Fn2 _Func);
```

## Remarks

This function behaves the same as the C++ Standard Library function [transform](#). For more information, see [transform](#).

## unique (STL/CLR)

Removes duplicate elements that are adjacent to each other in a specified range.

## Syntax

```
template<class _FwdIt> inline
    _FwdIt unique(_FwdIt _First, _FwdIt _Last);
template<class _FwdIt, class _Pr> inline
    _FwdIt unique(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [unique](#). For more information, see [unique](#).

## unique\_copy (STL/CLR)

Copies elements from a source range into a destination range except for the duplicate elements that are adjacent to each other.

## Syntax

```
template<class _InIt, class _OutIt> inline
    _OutIt unique_copy(_InIt _First, _InIt _Last, _OutIt _Dest);
template<class _InIt, class _OutIt, class _Pr> inline
    _OutIt unique_copy(_InIt _First, _InIt _Last, _OutIt _Dest,
        _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [unique\\_copy](#). For more information, see [unique\\_copy](#).

## upper\_bound (STL/CLR)

Finds the position of the first element in an ordered range that has a value that is greater than a specified value, where the ordering criterion may be specified by a binary predicate.

### Syntax

```
template<class _FwdIt, class _Ty> inline
    _FwdIt upper_bound(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);
template<class _FwdIt, class _Ty, class _Pr> inline
    _FwdIt upper_bound(_FwdIt _First, _FwdIt _Last,
        const _Ty% _Val, _Pr _Pred);
```

## Remarks

This function behaves the same as the C++ Standard Library function [upper\\_bound](#). For more information, see [upper\\_bound](#).

# deque (STL/CLR)

9/20/2022 • 37 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has random access. You use the container `deque` to manage a sequence of elements that looks like a contiguous block of storage, but which can grow or shrink at either end without the need to copy any remaining elements. Thus it can implement efficiently a `double-ended queue`. (Hence the name.)

In the description below, `GValue` is the same as `Value` unless the latter is a ref type, in which case it is `Value^`.

## Syntax

```
template<typename Value>
ref class deque
    : public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
System::Collections::Generic::IList<GValue>,
Microsoft::VisualC::StlClr::IDeque<GValue>
{ ..... };
```

## Parameters

### *GValue*

The generic type of an element in the controlled sequence.

### *Value*

The type of an element in the controlled sequence.

## Requirements

**Header:** <cliext/deque>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">deque::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.
<a href="#">deque::const_reference (STL/CLR)</a>	The type of a constant reference to an element.
<a href="#">deque::const_reverse_iterator (STL/CLR)</a>	The type of a constant reverse iterator for the controlled sequence.
<a href="#">deque::difference_type (STL/CLR)</a>	The type of a signed distance between two elements.
<a href="#">deque::generic_container (STL/CLR)</a>	The type of the generic interface for the container.

TYPE DEFINITION	DESCRIPTION
deque::generic_iterator (STL/CLR)	The type of an iterator for the generic interface for the container.
deque::generic_reverse_iterator (STL/CLR)	The type of a reverse iterator for the generic interface for the container.
deque::generic_value (STL/CLR)	The type of an element for the generic interface for the container.
deque::iterator (STL/CLR)	The type of an iterator for the controlled sequence.
deque::reference (STL/CLR)	The type of a reference to an element.
deque::reverse_iterator (STL/CLR)	The type of a reverse iterator for the controlled sequence.
deque::size_type (STL/CLR)	The type of a signed distance between two elements.
deque::value_type (STL/CLR)	The type of an element.

MEMBER FUNCTION	DESCRIPTION
deque::assign (STL/CLR)	Replaces all elements.
deque::at (STL/CLR)	Accesses an element at a specified position.
deque::back (STL/CLR)	Accesses the last element.
deque::begin (STL/CLR)	Designates the beginning of the controlled sequence.
deque::clear (STL/CLR)	Removes all elements.
deque::deque (STL/CLR)	Constructs a container object.
deque::empty (STL/CLR)	Tests whether no elements are present.
deque::end (STL/CLR)	Designates the end of the controlled sequence.
deque::erase (STL/CLR)	Removes elements at specified positions.
deque::front (STL/CLR)	Accesses the first element.
deque::insert (STL/CLR)	Adds elements at a specified position.
deque::pop_back (STL/CLR)	Removes the last element.
deque::pop_front (STL/CLR)	Removes the first element.
deque::push_back (STL/CLR)	Adds a new last element.
deque::push_front (STL/CLR)	Adds a new first element.

MEMBER FUNCTION	DESCRIPTION
deque::rbegin (STL/CLR)	Designates the beginning of the reversed controlled sequence.
deque::rend (STL/CLR)	Designates the end of the reversed controlled sequence.
deque::resize (STL/CLR)	Changes the number of elements.
deque::size (STL/CLR)	Counts the number of elements.
deque::swap (STL/CLR)	Swaps the contents of two containers.
deque::to_array (STL/CLR)	Copies the controlled sequence to a new array.

PROPERTY	DESCRIPTION
deque::back_item (STL/CLR)	Accesses the last element.
deque::front_item (STL/CLR)	Accesses the first element.

OPERATOR	DESCRIPTION
deque::operator!= (STL/CLR)	Determines if two <code>deque</code> objects are not equal.
deque::operator< (STL/CLR)	Accesses an element at a specified position.
operator< (deque) (STL/CLR)	Determines if a <code>deque</code> object is less than another <code>deque</code> object.
operator<= (deque) (STL/CLR)	Determines if a <code>deque</code> object is less than or equal to another <code>deque</code> object.
operator= (deque) (STL/CLR)	Replaces the controlled sequence.
operator== (deque) (STL/CLR)	Determines if a <code>deque</code> object is equal to another <code>deque</code> object.
operator> (deque) (STL/CLR)	Determines if a <code>deque</code> object is greater than another <code>deque</code> object.
operator>= (deque) (STL/CLR)	Determines if a <code>deque</code> object is greater than or equal to another <code>deque</code> object.

## Interfaces

INTERFACE	DESCRIPTION
ICloneable	Duplicate an object.
IEnumerable	Sequence through elements.

INTERFACE	DESCRIPTION
<a href="#">ICollection</a>	Maintain group of elements.
<a href="#">IEnumerable&lt;T&gt;</a>	Sequence through typed elements.
<a href="#">ICollection&lt;T&gt;</a>	Maintain group of typed elements.
<a href="#"> IList&lt;T&gt;</a>	Maintain ordered group of typed elements.
<a href="#">IDeque&lt;Value&gt;</a>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls through a stored array of handles that designate blocks of `Value` elements. The array grows on demand. Growth occurs in such a way that the cost of either prepending or appending a new element is constant time, and no remaining elements are disturbed. You can also remove an element at either end in constant time, and without disturbing remaining elements. Thus, a deque is a good candidate for the underlying container for template class [queue \(STL/CLR\)](#) or template class [stack \(STL/CLR\)](#).

A `deque` object supports random-access iterators, which means you can refer to an element directly given its numerical position, counting from zero for the first (front) element, to `deque::size (STL/CLR) () - 1` for the last (back) element. It also means that a deque is a good candidate for the underlying container for template class [priority\\_queue \(STL/CLR\)](#).

A deque iterator stores a handle to its associated deque object, along with the bias of the element it designates. You can use iterators only with their associated container objects. The bias of a deque element is *not* necessarily the same as its position. The first element inserted has bias zero, the next appended element has bias 1, but the next prepended element has bias -1.

Inserting or erasing elements at either end does *not* alter the value of an element stored at any valid bias. Inserting or erasing an interior element, however, *can* change the element value stored at a given bias, so the value designated by an iterator can also change. (The container may have to copy elements up or down to create a hole before an insert or to fill a hole after an erase.) Nevertheless, a deque iterator remains valid so long as its bias designates a valid element. Moreover, a valid iterator remains dereferencable -- you can use it to access or alter the element value it designates -- so long as its bias is not equal to the bias for the iterator returned by `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### `deque::assign (STL/CLR)`

Replaces all elements.

#### Syntax

```
void assign(size_type count, value_type val);
template<typename InIt>
    void assign(InIt first, InIt last);
void assign(System::Collections::Generic::IEnumarable<Value>^ right);
```

#### Parameters

*count*

Number of elements to insert.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Value of the element to insert.

#### Remarks

The first member function replaces the controlled sequence with a repetition of *count* elements of value *val*. You use it to fill the container with elements all having the same value.

If *InIt* is an integer type, the second member function behaves the same as

`assign((size_type)first, (value_type)last)`. Otherwise, it replaces the controlled sequence with the sequence [*first*, *last*]. You use it to make the controlled sequence a copy another sequence.

The third member function replaces the controlled sequence with the sequence designated by the enumerator *right*. You use it to make the controlled sequence a copy of a sequence described by an enumerator.

#### Example

```

// cliext_deque_assign.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // assign a repetition of values
    cliext::deque<wchar_t> c2;
    c2.assign(6, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an iterator range
    c2.assign(c1.begin(), c1.end() - 1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an enumeration
    c2.assign( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

x x x x x
a b
a b c

```

## deque::at (STL/CLR)

Accesses an element at a specified position.

### Syntax

```
reference at(size_type pos);
```

### Parameters

*pos*

Position of element to access.

### Remarks

The member function returns a reference to the element of the controlled sequence at position *pos*. You use it to read or write an element whose position you know.

### Example

```

// cliext_deque_at.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using at
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1.at(i));
    System::Console::WriteLine();

    // change an entry and redisplay
    c1.at(1) = L'x';
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a x c

```

## deque::back (STL/CLR)

Accesses the last element.

### Syntax

```
reference back();
```

### Remarks

The member function returns a reference to the last element of the controlled sequence, which must be non-empty. You use it to access the last element, when you know it exists.

### Example

```

// cliext_deque_back.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back() = {0}", c1.back());

    // alter last item and reinspect
    c1.back() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
back() = c
a b x

```

## deque::back\_item (STL/CLR)

Accesses the last element.

### Syntax

```
property value_type back_item;
```

### Remarks

The property accesses the last element of the controlled sequence, which must be non-empty. You use it to read or write the last element, when you know it exists.

### Example

```

// cliext_deque_back_item.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back_item = {0}", c1.back_item);

    // alter last item and reinspect
    c1.back_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
back_item = c
a b x

```

## deque::begin (STL/CLR)

Designates the beginning of the controlled sequence.

### Syntax

```

iterator begin();

```

### Remarks

The member function returns a random-access iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_deque_begin.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::deque<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*begin() = a
*++begin() = b
x y c

```

## deque::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

### Remarks

The member function effectively calls `deque::erase (STL/CLR) ( deque::begin (STL/CLR) (), deque::end (STL/CLR) () )`. You use it to ensure that the controlled sequence is empty.

### Example

```

// cliest deque_clear.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

a b c
size() = 0
a b
size() = 0

```

## deque::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

### Syntax

```

typedef T2 const_iterator;

```

### Remarks

The type describes an object of unspecified type `T2` that can serve as a constant random-access iterator for the controlled sequence.

### Example

```

// cliext_deque_const_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::deque<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## deque::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_deque_const_reference.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::deque<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        cliext::deque<wchar_t>::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## deque::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```
// cliext_deque_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::deque<wchar_t>::const_reverse_iterator crit = c1.rbegin();
    cliext::deque<wchar_t>::const_reverse_iterator crend = c1.rend();
    for (; crit != crend; ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## deque::deque (STL/CLR)

Constructs a container object.

### Syntax

```
deque();
deque(deque<Value>% right);
deque(deque<Value>^ right);
explicit deque(size_type count);
deque(size_type count, value_type val);
template<typename InIt>
    deque(InIt first, InIt last);
deque(System::Collections::Generic::IEnumerable<Value>^ right);
```

### Parameters

*count*

Number of elements to insert.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Object or range to insert.

*val*

Value of the element to insert.

## Remarks

The constructor:

```
deque();
```

initializes the controlled sequence with no elements. You use it to specify an empty initial controlled sequence.

The constructor:

```
deque(deque<Value>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`]. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the deque object *right*. For more information on the iterators, see [deque::begin \(STL/CLR\)](#) and [deque::end \(STL/CLR\)](#).

The constructor:

```
deque(deque<Value>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`]. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the deque object whose handle is *right*.

The constructor:

```
explicit deque(size_type count);
```

initializes the controlled sequence with *count* elements each with value `value_type()`. You use it to fill the container with elements all having the default value.

The constructor:

```
deque(size_type count, value_type val);
```

initializes the controlled sequence with *count* elements each with value *val*. You use it to fill the container with elements all having the same value.

The constructor:

```
template<typename InIt>
```

```
deque(InIt first, InIt last);
```

initializes the controlled sequence with the sequence [`first`, `last`]. You use it to make the controlled sequence a copy of another sequence.

The constructor:

```
deque(System::Collections::Generic::IEnumerable<Value>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*. You use it to make the controlled sequence a copy of another sequence described by an enumerator.

## Example

```

// cliext_deque_construct.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
// construct an empty container
    cliext::deque<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    // construct with a repetition of default values
    cliext::deque<wchar_t> c2(3);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

    // construct with a repetition of values
    cliext::deque<wchar_t> c3(6, L'x');
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an iterator range
    cliext::deque<wchar_t>::iterator it = c3.end();
    cliext::deque<wchar_t> c4(c3.begin(), --it);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an enumeration
    cliext::deque<wchar_t> c5(   // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
    for each (wchar_t elem in c5)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying another container
    cliext::deque<wchar_t> c7(c3);
    for each (wchar_t elem in c7)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying a container handle
    cliext::deque<wchar_t> c8(%c3);
    for each (wchar_t elem in c8)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}

```

```

size() = 0
0 0 0
x x x x x x
x x x x x
x x x x x x
x x x x x x
x x x x x x

```

## deque::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

## Remarks

The type describes a signed element count.

## Example

```
// cliext_deque_difference_type.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::deque<wchar_t>::difference_type diff = 0;
    for (cliext::deque<wchar_t>::iterator it = c1.begin();
         it != c1.end(); ++it) ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (cliext::deque<wchar_t>::iterator it = c1.end();
         it != c1.begin(); --it) --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
begin()-end() = -3
```

## deque::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

## Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [deque::size \(STL/CLR\)](#) `() == 0`. You use it to test whether the deque is empty.

## Example

```

// cliext_deque_empty.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

## deque::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a random-access iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the **current** end of the controlled sequence, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_deque_end.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    cliext::deque<wchar_t>::iterator it = c1.end();
    --it;
    System::Console::WriteLine("---- --end() = {0}", *--it);
    System::Console::WriteLine("----end() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
---- --end() = b
----end() = c
a x y

```

## deque::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);

```

### Parameters

*first*

Beginning of range to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [`first`, `last`]. You

use it to remove zero or more contiguous elements.

Both member functions return an iterator that designates the first element remaining beyond any elements removed, or `deque::end (STL/CLR)` if no such element exists.

When erasing elements, the number of element copies is linear in the number of elements between the end of the erasure and the nearer end of the sequence. (When erasing one or more elements at either end of the sequence, no element copies occur.)

## Example

```
// cliest deque_erase.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.push_back(L'd');
    c1.push_back(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    cliext::deque<wchar_t>::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

## deque::front (STL/CLR)

Accesses the first element.

### Syntax

```
reference front();
```

### Remarks

The member function returns a reference to the first element of the controlled sequence, which must be non-empty. You use it to read or write the first element, when you know it exists.

### Example

```
// cliext_deque_front.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front() = {0}", c1.front());

    // alter first item and reinspect
    c1.front() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front() = a
x b c
```

## deque::front\_item (STL/CLR)

Accesses the first element.

### Syntax

```
property value_type front_item;
```

### Remarks

The property accesses the first element of the controlled sequence, which must be non-empty. You use it to read or write the first element, when you know it exists.

### Example

```

// cliext_deque_front_item.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front_item = {0}", c1.front_item);

    // alter first item and reinspect
    c1.front_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
front_item = a
x b c

```

## deque::generic\_container (STL/CLR)

The type of the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::
    IDeque<generic_value>
generic_container;

```

### Remarks

The type describes the generic interface for this template container class.

### Example

```

// cliext deque_generic_container.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::deque<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(gc1->end(), L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push_back(L'e');

    System::Collections::IEnumerator^ enum1 =
        gc1->GetEnumerator();
    while (enum1->MoveNext())
        System::Console::Write("{0} ", enum1->Current);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

## deque::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerRandomAccessIterator<generic_value> generic_iterator;

```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_deque_generic_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::deque<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::deque<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::deque<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

## deque::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value> generic_reverse_iterator;

```

### Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_deque_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::deque<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::deque<wchar_t>::generic_reverse_iterator gcit = gc1->rbegin();
    cliext::deque<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a c c

```

## deque::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```

typedef GValue generic_value;

```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

### Example

```

// cliext_deque_generic_value.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::deque<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::deque<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::deque<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

## deque::insert (STL/CLR)

Adds elements at a specified position.

### Syntax

```

iterator insert(iterator where, value_type val);
void insert(iterator where, size_type count, value_type val);
template<typename InIt>
    void insert(iterator where, InIt first, InIt last);
void insert(iterator where,
    System::Collections::Generic::IEnumerable<Value>^ right);

```

### Parameters

*count*

Number of elements to insert.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Value of the element to insert.

*where*

Where in container to insert before.

## Remarks

Each of the member functions inserts, before the element pointed to by *where* in the controlled sequence, a sequence specified by the remaining operands.

The first member function inserts an element with value *val* and returns an iterator that designates the newly inserted element. You use it to insert a single element before a place designated by an iterator.

The second member function inserts a repetition of *count* elements of value *val*. You use it to insert zero or more contiguous elements which are all copies of the same value.

If `Init` is an integer type, the third member function behaves the same as

`insert(where, (size_type)first, (value_type)last)`. Otherwise, it inserts the sequence `[ first, last ]`. You use it to insert zero or more contiguous elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

When inserting a single element, the number of element copies is linear in the number of elements between the insertion point and the nearer end of the sequence. (When inserting one or more elements at either end of the sequence, no element copies occur.) If `Init` is an input iterator, the third member function effectively performs a single insertion for each element in the sequence. Otherwise, when inserting `N` elements, the number of element copies is linear in `N` plus the number of elements between the insertion point and the nearer end of the sequence.

## Example

```

// cliext_deque_insert.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value using iterator
    cliext::deque<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("insert(begin()+1, L'x') = {0}",
        *c1.insert(++it, L'x'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a repetition of values
    cliext::deque<wchar_t> c2;
    c2.insert(c2.begin(), 2, L'y');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    it = c1.end();
    c2.insert(c2.end(), c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    c2.insert(c2.begin(), // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(begin()+1, L'x') = x
a x b c
y y
y y a x b
a x b c y a x b

```

## deque::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

## Remarks

The type describes an object of unspecified type `T1` that can serve as a random-access iterator for the controlled sequence.

## Example

```
// cliext_deque_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::deque<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();

    // alter first element and redisplay
    it = c1.begin();
    *it = L'x';
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x b c
```

## deque::operator!= (STL/CLR)

Deque not equal comparison.

### Syntax

```
template<typename Value>
bool operator!=(deque<Value>% left,
                 deque<Value>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two deques are compared element by element.

## Example

```

// cliext_deque_operator_ne.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'c');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

## deque::operator(STL/CLR)

Accesses an element at a specified position.

### Syntax

```
reference operator[](size_type pos);
```

### Parameters

*pos*

Position of element to access.

### Remarks

The member operator returns a reference to the element at position *pos*. You use it to access an element whose position you know.

### Example

```

// cliext_deque_operator_sub.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using subscripting
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();

    // change an entry and redisplay
    c1[1] = L'x';
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a x c

```

## deque::pop\_back (STL/CLR)

Removes the last element.

### Syntax

```

void pop_back();

```

### Remarks

The member function removes the last element of the controlled sequence, which must be non-empty. You use it to shorten the deque by one element at the back.

### Example

```

// cliext_deque_pop_back.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_back();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b

```

## deque::pop\_front (STL/CLR)

Removes the first element.

### Syntax

```

void pop_front();

```

### Remarks

The member function removes the first element of the controlled sequence, which must be non-empty. You use it to shorten the deque by one element at the front.

### Example

```

// cliext_deque_pop_front.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_front();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
b c

```

## deque::push\_back (STL/CLR)

Adds a new last element.

### Syntax

```

void push_back(value_type val);

```

### Remarks

The member function inserts an element with value `val` at the end of the controlled sequence. You use it to append another element to the deque.

### Example

```

// cliext_deque_push_back.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

## deque::push\_front (STL/CLR)

Adds a new first element.

### Syntax

```
void push_front(value_type val);
```

### Remarks

The member function inserts an element with value `val` at the beginning of the controlled sequence. You use it to prepend another element to the deque.

### Example

```
// cliext_deque_push_front.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_front(L'a');
    c1.push_front(L'b');
    c1.push_front(L'c');

    // display contents " c b a"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## deque::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

### Syntax

```
reverse_iterator rbegin();
```

### Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the `beginning` of the reverse sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_deque_rbegin.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    cliext::deque<wchar_t>::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("++rbegin() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*rbegin() = c
*++rbegin() = b
a y x

```

## deque::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_deque_reference.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::deque<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        cliext::deque<wchar_t>::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();

    // modify contents " a b c"
    for (it = c1.begin(); it != c1.end(); ++it)
        { // get a reference to an element
        cliext::deque<wchar_t>::reference ref = *it;

        ref += (wchar_t)(L'A' - L'a');
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
A B C

```

## deque::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```

reverse_iterator rend();

```

### Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the **end** of the reverse sequence. You use it to obtain an iterator that designates the **current** end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_deque_rnd.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::deque<wchar_t>::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("---- --rend() = {0}", *--rit);
    System::Console::WriteLine("----rend() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
---- --rend() = b
----rend() = a
y x c

```

## deque::resize (STL/CLR)

Changes the number of elements.

### Syntax

```

void resize(size_type new_size);
void resize(size_type new_size, value_type val);

```

### Parameters

*new\_size*

New size of the controlled sequence.

*val*

Value of the padding element.

### Remarks

The member functions both ensure that [deque::size \(STL/CLR\)](#) () henceforth returns *new\_size*. If it must make the controlled sequence longer, the first member function appends elements with value [value\\_type\(\)](#), while the second member function appends elements with value *val*. To make the controlled sequence shorter, both member functions effectively erase the last element [deque::size \(STL/CLR\)](#) () - *new\_size* times. You use it to ensure that the controlled sequence has size *new\_size*, by either trimming or padding the current controlled sequence.

## Example

```
// cliext_deque_resize.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
// construct an empty container and pad with default values
    cliext::deque<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());
    c1.resize(4);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

    // resize to empty
    c1.resize(0);
    System::Console::WriteLine("size() = {0}", c1.size());

    // resize and pad
    c1.resize(5, L'x');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
size() = 0
0 0 0 0
size() = 0
x x x x x
```

## deque::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

### Syntax

```
typedef T3 reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

## Example

```

// cliest deque_reverse_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::deque<wchar_t>::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();

    // alter first element and redisplay
    rit = c1.rbegin();
    *rit = L'x';
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}

```

```

c b a
x b a

```

## deque::size (STL/CLR)

Counts the number of elements.

### Syntax

```

size_type size();

```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [deque::empty \(STL/CLR\)](#).

### Example

```

// cliext_deque_size.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

## deque::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_deque_size_type.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::deque<wchar_t>::size_type diff = c1.end() - c1.begin();
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3

```

## deque::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```

void swap(deque<Value>% right);

```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `*this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_deque_swap.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::deque<wchar_t> c2(5, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
x x x x x
x x x x x
a b c

```

## deque::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```
cli::array<Value>^ to_array();
```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_deque_to_array.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push_back(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c d
a b c

```

## deque::value\_type (STL/CLR)

The type of an element.

### Syntax

```

typedef Value value_type;

```

### Remarks

The type is a synonym for the template parameter *Value*.

### Example

```

// cliest deque_value_type.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using value_type
    for (cliext::deque<wchar_t>::iterator it = c1.begin();
        it != c1.end(); ++it)
    {   // store element in value_type object
        cliext::deque<wchar_t>::value_type val = *it;

        System::Console::Write("{0} ", val);
    }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## operator< (deque) (STL/CLR)

Deque less than comparison.

### Syntax

```

template<typename Value>
bool operator<(deque<Value>% left,
                 deque<Value>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true if, for the lowest position *i* for which  $!(right[i] < left[i])$  it is also true that  $left[i] < right[i]$ . Otherwise, it returns  $left->size() < right->size()$ . You use it to test whether *left* is ordered before *right* when the two deques are compared element by element.

### Example

```

// cliext_deque_operator_lt.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'c');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}

```

```

a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True

```

## operator<= (deque) (STL/CLR)

Deque less than or equal comparison.

### Syntax

```

template<typename Value>
bool operator<=(deque<Value>% left,
                 deque<Value>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two deques are compared element by element.

## Example

```
// cliext_deque_operator_le.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator= (deque) (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
deque<Value>% operator=(deque<Value>% right);
```

### Parameters

*right*

Container to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

## Example

```

// cliext_deque_operator_as.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

## operator== (deque) (STL/CLR)

Deque equal comparison.

### Syntax

```

template<typename Value>
bool operator==(deque<Value>% left,
                  deque<Value>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position *i*, *left[i] == right[i]*. You use it to test whether *left* is ordered the same as *right* when the two deques are compared element by element.

### Example

```

// cliext_deque_operator_eq.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}

```

```

a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False

```

## operator> (deque) (STL/CLR)

Deque greater than comparison.

### Syntax

```

template<typename Value>
bool operator>(deque<Value>% left,
                  deque<Value>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two deques are compared element by element.

## Example

```
// cliext_deque_operator_gt.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

## operator>= (deque) (STL/CLR)

Deque greater than or equal comparison.

### Syntax

```
template<typename Value>
bool operator>=(deque<Value>% left,
                 deque<Value>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right*.

when the two deques are compared element by element.

### Example

```
// cliext_deque_operator_ge.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
        c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

# functional (STL/CLR)

9/20/2022 • 34 minutes to read • [Edit Online](#)

Include the STL/CLR header `<cliext/functional>` to define the a number of template classes and related template delegates and functions.

## Syntax

```
#include <functional>
```

## Requirements

**Header:** `<cliext/functional>`

**Namespace:** `cliext`

## Declarations

DELEGATE	DESCRIPTION
<a href="#">binary_delegate (STL/CLR)</a>	Two-argument delegate.
<a href="#">binary_delegate_noreturn (STL/CLR)</a>	Two-argument delegate returning <code>void</code> .
<a href="#">unary_delegate (STL/CLR)</a>	One-argument delegate.
<a href="#">unary_delegate_noreturn (STL/CLR)</a>	One-argument delegate returning <code>void</code> .

CLASS	DESCRIPTION
<a href="#">binary_negate (STL/CLR)</a>	Functor to negate a two-argument functor.
<a href="#">binder1st (STL/CLR)</a>	Functor to bind first argument to a two-argument functor.
<a href="#">binder2nd (STL/CLR)</a>	Functor to bind second argument to a two-argument functor.
<a href="#">divides (STL/CLR)</a>	Divide functor.
<a href="#">equal_to (STL/CLR)</a>	Equal comparison functor.
<a href="#">greater (STL/CLR)</a>	Greater comparison functor.
<a href="#">greater_equal (STL/CLR)</a>	Greater or equal comparison functor.
<a href="#">less (STL/CLR)</a>	Less comparison functor.
<a href="#">less_equal (STL/CLR)</a>	Less or equal comparison functor.

CLASS	DESCRIPTION
logical_and (STL/CLR)	Logical AND functor.
logical_not (STL/CLR)	Logical NOT functor.
logical_or (STL/CLR)	Logical OR functor.
minus (STL/CLR)	Subtract functor.
modulus (STL/CLR)	Modulus functor.
multiples (STL/CLR)	Multiply functor.
negate (STL/CLR)	Functor to return its argument negated.
not_equal_to (STL/CLR)	Not equal comparison functor.
plus (STL/CLR)	Add functor.
unary_negate (STL/CLR)	Functor to negate a one-argument functor.
FUNCTION	DESCRIPTION
bind1st (STL/CLR)	Generates a binder1st for an argument and functor.
bind2nd (STL/CLR)	Generates a binder2nd for an argument and functor.
not1 (STL/CLR)	Generates a unary_negate for a functor.
not2 (STL/CLR)	Generates a binary_negate for a functor.

## Members

### binary\_delegate (STL/CLR)

The generic class describes a two-argument delegate. You use it specify a delegate in terms of its argument and return types.

#### Syntax

```
generic<typename Arg1,
        typename Arg2,
        typename Result>
delegate Result binary_delegate(Arg1, Arg2);
```

#### Parameters

##### Arg1

The type of the first argument.

##### Arg2

The type of the second argument.

*Result*

The return type.

## Remarks

The generic delegate describes a two-argument function.

Note that for:

```
binary_delegate<int, int, int> Fun1;
```

```
binary_delegate<int, int, int> Fun2;
```

the types `Fun1` and `Fun2` are synonyms, while for:

```
delegate int Fun1(int, int);
```

```
delegate int Fun2(int, int);
```

they are not the same type.

## Example

```
// cliext_binary_delegate.cpp
// compile with: /clr
#include <cliext/functional>

bool key_compare(wchar_t left, wchar_t right)
{
    return (left < right);
}

typedef cliext::binary_delegate<wchar_t, wchar_t, bool> Mydelegate;
int main()
{
    Mydelegate^ kcomp = gcnew Mydelegate(&key_compare);

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}
```

```
compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

## binary\_delegate\_noreturn (STL/CLR)

The generic class describes a two-argument delegate that returns `void`. You use it specify a delegate in terms of its argument.

### Syntax

```
generic<typename Arg1,
        typename Arg2>
    delegate void binary_delegate(Arg1, Arg2);
```

#### Parameters

*Arg1*

The type of the first argument.

*Arg2*

The type of the second argument.

#### Remarks

The generic delegate describes a two-argument function that returns `void`.

Note that for:

```
binary_delegate_noreturn<int, int> Fun1;
```

```
binary_delegate_noreturn<int, int> Fun2;
```

the types `Fun1` and `Fun2` are synonyms, while for:

```
delegate void Fun1(int, int);
```

```
delegate void Fun2(int, int);
```

they are not the same type.

#### Example

```
// cliext_binary_delegate_noreturn.cpp
// compile with: /clr
#include <cliext/functional>

void key_compare(wchar_t left, wchar_t right)
{
    System::Console::WriteLine("compare({0}, {1}) = {2}",
        left, right, left < right);
}

typedef cliext::binary_delegate_noreturn<wchar_t, wchar_t> Mydelegate;
int main()
{
    Mydelegate^ kcomp = gcnew Mydelegate(&key_compare);

    kcomp(L'a', L'a');
    kcomp(L'a', L'b');
    kcomp(L'b', L'a');
    System::Console::WriteLine();
    return (0);
}
```

```
compare(a, a) = False
compare(a, b) = True
compare(b, a) = False
```

## binary\_negate (STL/CLR)

The template class describes a functor that, when called, returns the logical NOT of its stored two-argument functor. You use it to specify a function object in terms of its stored functor.

## Syntax

```
template<typename Fun>
ref class binary_negate
{ // wrap operator()
public:
    typedef Fun stored_function_type;
    typedef typename Fun::first_argument_type first_argument_type;
    typedef typename Fun::second_argument_type second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
        delegate_type;

    explicit binary_negate(Fun% functor);
    binary_negate(binary_negate<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};
```

## Parameters

### *Fun*

The type of the stored functor.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.
stored_function_type	The type of the functor.

MEMBER	DESCRIPTION
binary_negate	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^()	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor that stores another two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns the logical NOT of the stored functor called with the two arguments.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted

appropriately.

## Example

```
// cliext_binary_negate.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::less<int> less_op;

    cliext::transform(c1.begin(), c1.begin() + 2,
                     c2.begin(), c3.begin(),
                     cliext::binary_negate<cliext::less<int>>(less_op));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2,
                     c2.begin(), c3.begin(), cliext::not2(less_op));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 3
4 4
1 0
1 0
```

## bind1st (STL/CLR)

Generates a `binder1st` for an argument and functor.

### Syntax

```
template<typename Fun,
         typename Arg>
binder1st<Fun> bind1st(Fun% functor,
                         Arg left);
```

#### Template Parameters

##### *Arg*

The type of the argument.

##### *Fun*

The type of the functor.

#### Function Parameters

##### *functor*

The functor to wrap.

##### *left*

The first argument to wrap.

#### Remarks

The template function returns [binder1st \(STL/CLR\)](#) `<Fun>(functor, left)`. You use it as a convenient way to wrap a two-argument functor and its first argument in a one-argument functor that calls it with a second argument.

#### Example

```
// cliext_bind1st.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::minus<int> sub_op;
    cliext::binder1st<cliext::minus<int> > subfrom3(sub_op, 3);

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                     subfrom3);
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                      bind1st(sub_op, 3));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 3
-1 0
-1 0
```

## bind2nd (STL/CLR)

Generates a `binder2nd` for an argument and functor.

### Syntax

```
template<typename Fun,
         typename Arg>
binder2nd<Fun> bind2nd(Fun% functor,
                         Arg right);
```

#### Template Parameters

*Arg*

The type of the argument.

*Fun*

The type of the functor.

#### Function Parameters

*functor*

The functor to wrap.

*right*

The second argument to wrap.

### Remarks

The template function returns `binder2nd (STL/CLR) <Fun>(functor, right)`. You use it as a convenient way to wrap a two-argument functor and its second argument in a one-argument functor that calls it with a first argument.

### Example

```

// cliext_bind2nd.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::minus<int> sub_op;
    cliext::binder2nd<cliext::minus<int> > sub4(sub_op, 4);

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                     sub4);
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                      bind2nd(sub_op, 4));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
0 -1
0 -1

```

## binder1st (STL/CLR)

The template class describes a one-argument functor that, when called, returns its stored two-argument functor called with its stored first argument and the supplied second argument. You use it specify a function object in terms of its stored functor.

### Syntax

```

template<typename Fun>
ref class binder1st
{ // wrap operator()
public:
    typedef Fun stored_function_type;
    typedef typename Fun::first_argument_type first_argument_type;
    typedef typename Fun::second_argument_type second_argument_type;
    typedef typename Fun::result_type result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        second_argument_type, result_type>
    delegate_type;

    binder1st(Fun% functor, first_argument_type left);
    binder1st(binder1st<Arg>% right);

    result_type operator()(second_argument_type right);
    operator delegate_type^();
};


```

## Parameters

### *Fun*

The type of the stored functor.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.
stored_function_type	The type of the functor.
MEMBER	DESCRIPTION
binder1st	Constructs the functor.
OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^()	Casts the functor to a delegate.

## Remarks

The template class describes a one-argument functor that stores a two-argument functor and a first argument. It defines the member operator `operator()` so that, when the object is called as a function, it returns the result of calling the stored functor with the stored first argument and the supplied second argument.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_binder1st.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::minus<int> sub_op;
    cliext::binder1st<cliext::minus<int> > subfrom3(sub_op, 3);

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                     subfrom3);
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                      bind1st(sub_op, 3));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
-1 0
-1 0

```

## binder2nd (STL/CLR)

The template class describes a one-argument functor that, when called, returns its stored two-argument functor called with the supplied first argument and its stored second argument. You use it specify a function object in terms of its stored functor.

### Syntax

```

template<typename Fun>
ref class binder2nd
{ // wrap operator()
public:
    typedef Fun stored_function_type;
    typedef typename Fun::first_argument_type first_argument_type;
    typedef typename Fun::second_argument_type second_argument_type;
    typedef typename Fun::result_type result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        first_argument_type, result_type>
        delegate_type;

    binder2nd(Fun% functor, second_argument_type left);
    binder2nd(binder2nd<Arg>% right);

    result_type operator()(first_argument_type right);
    operator delegate_type^();
};


```

## Parameters

### *Fun*

The type of the stored functor.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.
stored_function_type	The type of the functor.

  

MEMBER	DESCRIPTION
binder2nd	Constructs the functor.

  

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^()	Casts the functor to a delegate.

## Remarks

The template class describes a one-argument functor that stores a two-argument functor and a second argument. It defines the member operator `operator()` so that, when the object is called as a function, it returns the result of calling the stored functor with the supplied first argument and the stored second argument.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_binder2nd.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::minus<int> sub_op;
    cliext::binder2nd<cliext::minus<int> > sub4(sub_op, 4);

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                     sub4);
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                      bind2nd(sub_op, 4));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
0 -1
0 -1

```

## divides (STL/CLR)

The template class describes a functor that, when called, returns the first argument divided by the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class divides
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    divides();
    divides(divides<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments and return value.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
divides	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^()	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns the first argument divided by the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_divides.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(2);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 2 1"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::divides<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
2 1
2 3

```

## equal\_to (STL/CLR)

The template class describes a functor that, when called, returns true only if the first argument is equal to the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class equal_to
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    equal_to();
    equal_to(equal_to<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

  

MEMBER	DESCRIPTION
equal_to	Constructs the functor.

  

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^()	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if the first argument is equal to the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_equal_to.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::equal_to<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
4 4
1 0

```

## greater (STL/CLR)

The template class describes a functor that, when called, returns true only if the first argument is greater than the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class greater
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    greater();
    greater(greater<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
greater	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if the first argument is greater than the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_greater.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(3);
    c2.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 3 3"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::greater<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
3 3
1 0

```

## greater\_equal (STL/CLR)

The template class describes a functor that, when called, returns true only if the first argument is greater than or equal to the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class greater_equal
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    greater_equal();
    greater_equal(greater_equal<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

```

## Parameters

### *Arg*

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
greater_equal	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if the first argument is greater than or equal to the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliest_greater_equal.cpp
// compile with: /clr
#include <cliest/algorithm>
#include <cliest/functional>
#include <cliest/vector>

typedef cliest::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliest::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliest::greater_equal<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
4 4
1 0

```

## less (STL/CLR)

The template class describes a functor that, when called, returns true only if the first argument is less than the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class less
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    less();
    less(less<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
less	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if the first argument is less than the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_less.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::less<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
4 4
0 1

```

## less\_equal (STL/CLR)

The template class describes a functor that, when called, returns true only if the first argument is less than or equal to the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class less_equal
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    less_equal();
    less_equal(less_equal<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
less_equal	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if the first argument is less than or equal to the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliest_less_equal.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliest::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(3);
    c2.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 3 3"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliest::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliest::less_equal<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
3 3
0 1

```

## logical\_and (STL/CLR)

The template class describes a functor that, when called, returns true only if both the first argument and the second test as true. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class logical_and
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    logical_and();
    logical_and(logical_and<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

```

## Parameters

### *Arg*

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
logical_and	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if both the first argument and the second test as true.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_logical_and.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(2);
    c1.push_back(0);
    Myvector c2;
    c2.push_back(3);
    c2.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 1 0" and " 1 0"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::logical_and<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

2 0
3 0
1 0

```

## logical\_not (STL/CLR)

The template class describes a functor that, when called, returns true only if either its argument tests as false. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class logical_not
{ // wrap operator()
public:
    typedef Arg argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        argument_type, result_type>
        delegate_type;

    logical_not();
    logical_not(logical_not<Arg> %right);

    result_type operator()(argument_type left);
    operator delegate_type^();
};

```

## Parameters

### Arg

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
argument_type	The type of the functor argument.
delegate_type	The type of the generic delegate.
result_type	The type of the functor result.
MEMBER	DESCRIPTION
logical_not	Constructs the functor.
OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a one-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if its argument tests as false.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_logical_not.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 4 0"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c3.begin(), cliext::logical_not<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 0
0 1

```

## logical\_or (STL/CLR)

The template class describes a functor that, when called, returns true only if either the first argument or the second tests as true. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class logical_or
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
        delegate_type;

    logical_or();
    logical_or(logical_or<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

```

### Parameters

#### *Arg*

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.
MEMBER	DESCRIPTION
logical_or	Constructs the functor.
OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if either the first argument or the second tests as true.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_logical_or.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(2);
    c1.push_back(0);
    Myvector c2;
    c2.push_back(0);
    c2.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 2 0" and " 0 0"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::logical_or<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

2 0
0 0
1 0

```

## minus (STL/CLR)

The template class describes a functor that, when called, returns the first argument minus the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class minus
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    minus();
    minus(minus<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments and return value.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
minus	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns the first argument minus the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_minus.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(2);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 2 1"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::minus<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
2 1
2 2

```

## modulus (STL/CLR)

The template class describes a functor that, when called, returns the first argument modulo the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class modulus
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    modulus();
    modulus(modulus<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments and return value.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
modulus	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns the first argument modulo the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_modulus.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(2);
    Myvector c2;
    c2.push_back(3);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 2" and " 3 1"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::modulus<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 2
3 1
1 0

```

## multiplies (STL/CLR)

The template class describes a functor that, when called, returns the first argument times the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class multiplies
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    multiplies();
    multiplies(multiplies<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments and return value.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
multiplies	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns the first argument times the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_multiplies.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(2);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 2 1"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::multiplies<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
2 1
8 3

```

## negate (STL/CLR)

The template class describes a functor that, when called, returns its argument negated. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class negate
{ // wrap operator()
public:
    typedef Arg argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        argument_type, result_type>
        delegate_type;

    negate();
    negate(negate<Arg>% right);

    result_type operator()(argument_type left);
    operator delegate_type^();
};

```

## Parameters

### Arg

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
argument_type	The type of the functor argument.
delegate_type	The type of the generic delegate.
result_type	The type of the functor result.
MEMBER	DESCRIPTION
negate	Constructs the functor.
OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a one-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns its argument negated.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_negate.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(-3);
    Myvector c3(2, 0);

    // display initial contents " 4 -3"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c3.begin(), cliext::negate<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 -3
-4 3

```

## not\_equal\_to (STL/CLR)

The template class describes a functor that, when called, returns true only if the first argument is not equal to the second. You use it specify a function object in terms of its argument type.

### Syntax

```

template<typename Arg>
ref class not_equal_to
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
        delegate_type;

    not_equal_to();
    not_equal_to(not_equal_to<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

```

### Parameters

#### *Arg*

The type of the arguments.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.
MEMBER	DESCRIPTION
not_equal_to	Constructs the functor.
OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns true only if the first argument is not equal to the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_not_equal_to.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::not_equal_to<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
4 4
0 1

```

## not1 (STL/CLR)

Generates a `unary_negate` for a functor.

### Syntax

```

template<typename Fun>
unary_negate<Fun> not1(Fun% functor);

```

#### Template Parameters

*Fun*

The type of the functor.

#### Function Parameters

*functor*

The functor to wrap.

### Remarks

The template function returns `unary_negate (STL/CLR) <Fun>(functor)`. You use it as a convenient way to wrap a one-argument functor in a functor that delivers its logical NOT.

## Example

```
// cliext_not1.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 4 0"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::logical_not<int> not_op;

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        cliext::unary_negate<cliext::logical_not<int>>(not_op));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        cliext::not1(not_op));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 0
1 0
1 0
```

## not2 (STL/CLR)

Generates a `binary_negate` for a functor.

### Syntax

```
template<typename Fun>
binary_negate<Fun> not2(Fun% functor);
```

#### Template Parameters

*Fun*

The type of the functor.

#### Function Parameters

*functor*

The functor to wrap.

### Remarks

The template function returns [binary\\_negate \(STL/CLR\)](#) `<Fun>(functor)`. You use it as a convenient way to wrap a two-argument functor in a functor that delivers its logical NOT.

## Example

```
// cliext_not2.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::less<int> less_op;

    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(),
                      cliext::binary_negate<cliext::less<int> >(less_op));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::not2(less_op));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 3
4 4
1 0
1 0
```

## plus (STL/CLR)

The template class describes a functor that, when called, returns the first argument plus the second. You use it to specify a function object in terms of its argument type.

## Syntax

```

template<typename Arg>
ref class plus
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    plus();
    plus(plus<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

}

```

## Parameters

### *Arg*

The type of the arguments and return value.

## Member Functions

TYPE DEFINITION	DESCRIPTION
delegate_type	The type of the generic delegate.
first_argument_type	The type of the functor first argument.
result_type	The type of the functor result.
second_argument_type	The type of the functor second argument.

MEMBER	DESCRIPTION
plus	Constructs the functor.

OPERATOR	DESCRIPTION
operator()	Computes the desired function.
operator delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a two-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns the first argument plus the second.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_plus.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(2);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 2 1"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::plus<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
2 1
6 4

```

## unary\_delegate (STL/CLR)

The generic class describes a one-argument delegate. You use it specify a delegate in terms of its argument and return types.

### Syntax

```

generic<typename Arg,
        typename Result>
delegate Result unary_delegate(Arg);

```

### Parameters

*Arg*

The type of the argument.

*Result*

The return type.

### Remarks

The generic delegate describes a one-argument function.

Note that for:

```
unary_delegate<int, int> Fun1;
```

```
unary_delegate<int, int> Fun2;
```

the types `Fun1` and `Fun2` are synonyms, while for:

```
delegate int Fun1(int);
```

```
delegate int Fun2(int);
```

they are not the same type.

## Example

```
// cliext_unary_delegate.cpp
// compile with: /clr
#include <cliext/functional>

int hash_val(wchar_t val)
{
    return ((val * 17 + 31) % 67);

typedef cliext::unary_delegate<wchar_t, int> Mydelegate;
int main()
{
    Mydelegate^ myhash = gcnew Mydelegate(&hash_val);

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 5
hash(L'b') = 22
```

## unary\_delegate\_noreturn (STL/CLR)

The generic class describes a one-argument delegate that returns `void`. You use it specify a delegate in terms of its argument type.

### Syntax

```
generic<typename Arg>
delegate void unary_delegate_noreturn(Arg);
```

### Parameters

*Arg*

The type of the argument.

### Remarks

The generic delegate describes a one-argument function that returns `void`.

Note that for:

```
unary_delegate_noreturn<int> Fun1;
```

```
unary_delegate_noreturn<int> Fun2;
```

the types `Fun1` and `Fun2` are synonyms, while for:

```
delegate void Fun1(int);
```

```
delegate void Fun2(int);
```

they are not the same type.

## Example

```
// cliext_unary_delegate_noreturn.cpp
// compile with: /clr
#include <cliext/functional>

void hash_val(wchar_t val)
{
    System::Console::WriteLine("hash({0}) = {1}",
        val, (val * 17 + 31) % 67);
}

typedef cliext::unary_delegate_noreturn<wchar_t> Mydelegate;
int main()
{
    Mydelegate^ myhash = gcnew Mydelegate(&hash_val);

    myhash(L'a');
    myhash(L'b');
    return (0);
}
```

```
hash(a) = 5
hash(b) = 22
```

## unary\_negate (STL/CLR)

The template class describes a functor that, when called, returns the logical NOT of its stored one-argument functor. You use it specify a function object in terms of its stored functor.

### Syntax

```
template<typename Fun>
ref class unary_negate
{ // wrap operator()
public:
    typedef Fun stored_function_type;
    typedef typename Fun::argument_type argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        argument_type, result_type>
        delegate_type;

    unary_negate(Fun% functor);
    unary_negate(unary_negate<Fun>% right);

    result_type operator()(argument_type left);
    operator delegate_type^();
};
```

### Parameters

*Fun*

The type of the stored functor.

## Member Functions

TYPE DEFINITION	DESCRIPTION
argument_type	The type of the functor argument.
delegate_type	The type of the generic delegate.
result_type	The type of the functor result.
MEMBER	DESCRIPTION
unary_negate	Constructs the functor.
OPERATOR	DESCRIPTION
operator()	Computes the desired function.
delegate_type^	Casts the functor to a delegate.

## Remarks

The template class describes a one-argument functor that stores another one-argument functor. It defines the member operator `operator()` so that, when the object is called as a function, it returns the logical NOT of the stored functor called with the argument.

You can also pass the object as a function argument whose type is `delegate_type^` and it will be converted appropriately.

## Example

```

// cliext_unary_negate.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 4 0"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::logical_not<int> not_op;

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        cliext::unary_negate<cliext::logical_not<int> >(not_op));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        cliext::not1(not_op));
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 0
1 0
1 0

```

# hash\_map (STL/CLR)

9/20/2022 • 44 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `hash_map` to manage a sequence of elements as a hash table, each table entry storing a bidirectional linked list of nodes, and each node storing one element. An element consists of a key, for ordering the sequence, and a mapped value, which goes along for the ride.

In the description below, `GValue` is the same as:

`Microsoft::VisualC::StlClr::GenericPair<GKey, GMapped>`

where:

`GKey` is the same as `Key` unless the latter is a ref type, in which case it is `Key^`

`GMapped` is the same as `Mapped` unless the latter is a ref type, in which case it is `Mapped^`

## Syntax

```
template<typename Key,
         typename Mapped>
ref class hash_map
    : public
        System::ICloneable,
        System::Collections::IEnumerable,
        System::Collections::ICollection,
        System::Collections::Generic::IEnumerable<GValue>,
        System::Collections::Generic::ICollection<GValue>,
        System::Collections::Generic::IList<GValue>,
        System::Collections::Generic::IDictionary<Gkey, GMapped>,
        Microsoft::VisualC::StlClr::IHash<Gkey, GValue>
{ ..... };
```

## Parameters

### *Key*

The type of the key component of an element in the controlled sequence.

### *Mapped*

The type of the additional component of an element in the controlled sequence.

## Requirements

**Header:** <cliext/hash\_map>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">hash_map::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.

TYPE DEFINITION	DESCRIPTION
<code>hash_map::const_reference (STL/CLR)</code>	The type of a constant reference to an element.
<code>hash_map::const_reverse_iterator (STL/CLR)</code>	The type of a constant reverse iterator for the controlled sequence.
<code>hash_map::difference_type (STL/CLR)</code>	The type of a (possibly signed) distance between two elements.
<code>hash_map::generic_container (STL/CLR)</code>	The type of the generic interface for the container.
<code>hash_map::generic_iterator (STL/CLR)</code>	The type of an iterator for the generic interface for the container.
<code>hash_map::generic_reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the generic interface for the container.
<code>hash_map::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container.
<code>hash_map::hasher (STL/CLR)</code>	The hashing delegate for a key.
<code>hash_map::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>hash_map::key_compare (STL/CLR)</code>	The ordering delegate for two keys.
<code>hash_map::key_type (STL/CLR)</code>	The type of an ordering key.
<code>hash_map::mapped_type (STL/CLR)</code>	The type of the mapped value associated with each key.
<code>hash_map::reference (STL/CLR)</code>	The type of a reference to an element.
<code>hash_map::reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the controlled sequence.
<code>hash_map::size_type (STL/CLR)</code>	The type of a (non-negative) distance between two elements.
<code>hash_map::value_compare (STL/CLR)</code>	The ordering delegate for two element values.
<code>hash_map::value_type (STL/CLR)</code>	The type of an element.

MEMBER FUNCTION	DESCRIPTION
<code>hash_map::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>hash_map::bucket_count (STL/CLR)</code>	Counts the number of buckets.
<code>hash_map::clear (STL/CLR)</code>	Removes all elements.
<code>hash_map::count (STL/CLR)</code>	Counts elements matching a specified key.
<code>hash_map::empty (STL/CLR)</code>	Tests whether no elements are present.

MEMBER FUNCTION	DESCRIPTION
<code>hash_map::end (STL/CLR)</code>	Designates the end of the controlled sequence.
<code>hash_map::equal_range (STL/CLR)</code>	Finds range that matches a specified key.
<code>hash_map::erase (STL/CLR)</code>	Removes elements at specified positions.
<code>hash_map::find (STL/CLR)</code>	Finds an element that matches a specified key.
<code>hash_map::hash_delegate (STL/CLR)</code>	Copies the hashing delegate for a key.
<code>hash_map::hash_map (STL/CLR)</code>	Constructs a container object.
<code>hash_map::insert (STL/CLR)</code>	Adds elements.
<code>hash_map::key_comp (STL/CLR)</code>	Copies the ordering delegate for two keys.
<code>hash_map::load_factor (STL/CLR)</code>	Counts the average elements per bucket.
<code>hash_map::lower_bound (STL/CLR)</code>	Finds beginning of range that matches a specified key.
<code>hash_map::make_value (STL/CLR)</code>	Constructs a value object.
<code>hash_map::max_load_factor (STL/CLR)</code>	Gets or sets the maximum elements per bucket.
<code>hash_map::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.
<code>hash_map::rehash (STL/CLR)</code>	Rebuilds the hash table.
<code>hash_map::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>hash_map::size (STL/CLR)</code>	Counts the number of elements.
<code>hash_map::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>hash_map::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>hash_map::upper_bound (STL/CLR)</code>	Finds end of range that matches a specified key.
<code>hash_map::value_comp (STL/CLR)</code>	Copies the ordering delegate for two element values.
OPERATOR	DESCRIPTION
<code>hash_map::operator= (STL/CLR)</code>	Replaces the controlled sequence.
<code>hash_map::operator(STL/CLR)</code>	Maps a key to its associated mapped value.

## Interfaces

INTERFACE	DESCRIPTION
<a href="#">ICloneable</a>	Duplicate an object.
<a href="#">IEnumerable</a>	Sequence through elements.
<a href="#">ICollection</a>	Maintain group of elements.
<a href="#">IEnumerable&lt;T&gt;</a>	Sequence through typed elements.
<a href="#">ICollection&lt;T&gt;</a>	Maintain group of typed elements.
<a href="#">IDictionary&lt; TKey, TValue &gt;</a>	Maintain group of {key, value} pairs.
<a href="#">IHash&lt; Key, Value &gt;</a>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes in a bidirectional linked list. To speed access, the object also maintains a varying-length array of pointers into the list (the hash table), effectively managing the whole list as a sequence of sublists, or buckets. It inserts elements into a bucket that it keeps ordered by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements.

The object orders each bucket it controls by calling a stored delegate object of type [hash\\_set::key\\_compare \(STL/CLR\)](#). You can specify the stored delegate object when you construct the `hash_set`; if you specify no delegate object, the default is the comparison `operator<=(key_type, key_type)`.

You access the stored delegate object by calling the member function [hash\\_set::key\\_comp \(STL/CLR\)](#). Such a delegate object must define equivalent ordering between keys of type [hash\\_set::key\\_type \(STL/CLR\)](#). That means, for any two keys `x` and `y`:

`key_comp()(x, y)` returns the same Boolean result on every call.

If `key_comp()(x, y) && key_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

Any ordering rule that behaves like `operator<=(key_type, key_type)`, `operator>=(key_type, key_type)` or `operator==(key_type, key_type)` defines equivalent ordering.

Note that the container ensures only that elements whose keys have equivalent ordering (and which hash to the same integer value) are adjacent within a bucket. Unlike template class [hash\\_multimap \(STL/CLR\)](#), an object of template class `hash_map` ensures that keys for all elements are unique. (No two keys have equivalent ordering.)

The object determines which bucket should contain a given ordering key by calling a stored delegate object of type [hash\\_set::hasher \(STL/CLR\)](#). You access this stored object by calling the member function [hash\\_set::hash\\_delegate \(STL/CLR\)](#) to obtain an integer value that depends on the key value. You can specify the stored delegate object when you construct the `hash_set`; if you specify no delegate object, the default is the function `System::Object::hash_value(key_type)`. That means, for any keys `x` and `y`:

`hash_delegate()(x)` returns the same integer result on every call.

If `x` and `y` have equivalent ordering, then `hash_delegate()(x)` should return the same integer result as `hash_delegate()(y)`.

Each element contains a separate key and a mapped value. The sequence is represented in a way that permits lookup, insertion, and removal of an arbitrary element with a number of operations that is independent of the

number of elements in the sequence (constant time) -- at least in the best of cases. Moreover, inserting an element invalidates no iterators, and removing an element invalidates only those iterators which point at the removed element.

If hashed values are not uniformly distributed, however, a hash table can degenerate. In the extreme -- for a hash function that always returns the same value -- lookup, insertion, and removal are proportional to the number of elements in the sequence (linear time). The container endeavors to choose a reasonable hash function, mean bucket size, and hash-table size (total number of buckets), but you can override any or all of these choices. See, for example, the functions [hash\\_set::max\\_load\\_factor \(STL/CLR\)](#) and [hash\\_set::rehash \(STL/CLR\)](#).

A `hash_map` supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by [hash\\_map::end \(STL/CLR\)](#). You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a `hash_map` iterator to reach the head node, and it will then compare equal to `end()`. But you cannot dereference the iterator returned by `end()`.

Note that you cannot refer to a `hash_map` element directly given its numerical position -- that requires a random-access iterator.

A `hash_map` iterator stores a handle to its associated `hash_map` node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A `hash_map` iterator remains valid so long as its associated `hash_map` node is associated with some `hash_map`. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### `hash_map::begin (STL/CLR)`

Designates the beginning of the controlled sequence.

#### Syntax

```
iterator begin();
```

#### Remarks

The member function returns a bidirectional iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

#### Example

```

// cliext_hash_map_begin.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_map::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*++begin() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*begin() = [a 1]
*++begin() = [b 2]

```

## hash\_map::bucket\_count (STL/CLR)

Counts the number of buckets.

### Syntax

```
int bucket_count();
```

### Remarks

The member functions returns the current number of buckets. You use it to determine the size of the hash table.

### Example

```

// cliext_hash_map_bucket_count.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1 = gcnew Myhash_map;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_map::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

## Remarks

The member function effectively calls `hash_map::erase (STL/CLR)()`, `hash_map::begin (STL/CLR)()`, `hash_map::end (STL/CLR)()`. You use it to ensure that the controlled sequence is empty.

## Example

```
// cliext_hash_map_clear.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));

    // display contents " [a 1] [b 2]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2]
size() = 0
```

## hash\_map::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

## Syntax

```
typedef T2 const_iterator;
```

## Remarks

The type describes an object of unspecified type `T2` that can serve as a constant bidirectional iterator for the controlled sequence.

## Example

```

// cliext_hash_map_const_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_map::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("[{0} {1}] ", cit->first, cit->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## hash\_map::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_hash_map_const_reference.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_map::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
    {
        // get a const reference to an element
        Myhash_map::const_reference cref = *cit;
        System::Console::Write("[{0} {1}] ", cref->first, cref->second);
    }
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
```

## hash\_map::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```
// cliext_hash_map_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Myhash_map::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("[{0} {1}] ", crit->first, crit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

## hash\_map::count (STL/CLR)

Finds the number of elements matching a specified key.

### Syntax

```
size_type count(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns the number of elements in the controlled sequence that have equivalent ordering with *key*. You use it to determine the number of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_hash_map_count.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents "[a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

## hash\_map::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a possibly negative element count.

## Example

```

// cliext_hash_map_difference_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_map::difference_type diff = 0;
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Myhash_map::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3
begin()-end() = -3

```

## hash\_map::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

### Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [hash\\_map::size \(STL/CLR\) \(\) == 0](#). You use it to test whether the hash\_map is empty.

### Example

```

// cliext_hash_map_empty.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 3
empty() = False
size() = 0
empty() = True

```

## hash\_map::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a bidirectional iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_hash_map_end.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect last two items
    Myhash_map::iterator it = c1.end();
    --it;
    --it;
    System::Console::WriteLine("--- --end() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("---end() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
--- --end() = [b 2]
---end() = [c 3]

```

## hash\_map::equal\_range (STL/CLR)

Finds range that matches a specified key.

### Syntax

```
cliext::pair<iterator, iterator> equal_range(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns a pair of iterators

`cliext::pair<iterator, iterator>(lower_bound(key), upper_bound(key))`. You use it to determine the range of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_hash_map_equal_range.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
typedef Myhash_map::pair_iter_iter Pairii;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("[{0} {1}] ",
            pair1.first->first, pair1.first->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
equal_range(L'x') empty = True
[b 2]

```

## hash\_map::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

### Parameters

*first*

Beginning of range to erase.

*key*

Key value to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*, and returns an iterator that designates the first element remaining beyond the element removed, or [hash\\_map::end \(STL/CLR\)](#) if no such element exists. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [`first`, `last`), and returns an iterator that designates the first element remaining beyond any elements removed, or `end()` if no such element exists.. You use it to remove zero or more contiguous elements.

The third member function removes any element of the controlled sequence whose key has equivalent ordering to *key*, and returns a count of the number of elements removed. You use it to remove and count all elements that match a specified key.

Each element erasure takes time proportional to the logarithm of the number of elements in the controlled sequence.

## Example

```
// cliext_hash_map_erase.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    cliext::hash_map<wchar_t, int> c1;
    c1.insert(cliext::hash_map<wchar_t, int>::value_type(L'a', 1));
    c1.insert(cliext::hash_map<wchar_t, int>::value_type(L'b', 2));
    c1.insert(cliext::hash_map<wchar_t, int>::value_type(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (cliext::hash_map<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase an element and reinspect
    cliext::hash_map<wchar_t, int>::iterator it =
        c1.erase(c1.begin());
    System::Console::WriteLine("erase(begin()) = [{0} {1}]",
        it->first, it->second);

    // add elements and display " b c d e"
    c1.insert(cliext::hash_map<wchar_t, int>::value_type(L'd', 4));
    c1.insert(cliext::hash_map<wchar_t, int>::value_type(L'e', 5));
    for each (cliext::hash_map<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase all but end
    it = c1.end();
    it = c1.erase(c1.begin(), --it);
    System::Console::WriteLine("erase(begin(), end()-1) = [{0} {1}]",
        it->first, it->second);
    System::Console::WriteLine("size() = {0}", c1.size());

    // erase end
    System::Console::WriteLine("erase(L'x') = {0}", c1.erase(L'x'));
    System::Console::WriteLine("erase(L'e') = {0}", c1.erase(L'e'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
erase(begin()) = [b 2]
[b 2] [c 3] [d 4] [e 5]
erase(begin(), end()-1) = [e 5]
size() = 1
erase(L'x') = 0
erase(L'e') = 1
```

## hash\_map::find (STL/CLR)

Finds an element that matches a specified key.

### Syntax

```
iterator find(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

If at least one element in the controlled sequence has equivalent ordering with *key*, the member function returns an iterator designating one of those elements; otherwise it returns [hash\\_map::end \(STL/CLR\)](#) (). You use it to locate an element currently in the controlled sequence that matches a specified key.

### Example

```
// cliext_hash_map_find.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());

    Myhash_map::iterator it = c1.find(L'b');
    System::Console::WriteLine("find {0} = [{1} {2}]",
        L'b', it->first, it->second);

    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
find A = False
find b = [b 2]
find C = False
```

## hash\_map::generic\_container (STL/CLR)

The type of the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::
    IHash<GKey, GValue>
generic_container;
```

### Remarks

The type describes the generic interface for this template container class.

### Example

```
// cliext_hash_map_generic_container.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_map::generic_container^ gc1 = %c1;
    for each (Myhash_map::value_type elem in gc1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(Myhash_map::make_value(L'd', 4));
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(Myhash_map::make_value(L'e', 5));
    for each (Myhash_map::value_type elem in gc1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3] [d 4] [e 5]
```

## hash\_map::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

### Example

```
// cliext_hash_map_generic_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_map::generic_container^ gc1 = %c1;
    for each (Myhash_map::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_map::generic_iterator gcit = gc1->begin();
    Myhash_map::generic_value gcval = *gcit;
    System::Console::Write("[{0} {1}] ", gcval->first, gcval->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]
```

## hash\_map::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::  
    ReverseRandomAccessIterator<generic_value>  
    generic_reverse_iterator;
```

## Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

## Example

```
// cliext_hash_map_generic_reverse_iterator.cpp  
// compile with: /clr  
#include <cliext/hash_map>  
  
typedef cliext::hash_map<wchar_t, int> Myhash_map;  
int main()  
{  
    Myhash_map c1;  
    c1.insert(Myhash_map::make_value(L'a', 1));  
    c1.insert(Myhash_map::make_value(L'b', 2));  
    c1.insert(Myhash_map::make_value(L'c', 3));  
  
    // display contents " [a 1] [b 2] [c 3]"  
    for each (Myhash_map::value_type elem in c1)  
        System::Console::Write("{0} {1}]", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // construct a generic container  
    Myhash_map::generic_container^ gc1 = %c1;  
    for each (Myhash_map::value_type elem in gc1)  
        System::Console::Write("{0} {1}]", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // get an element and display it  
    Myhash_map::generic_reverse_iterator gcit = gc1->rbegin();  
    Myhash_map::generic_value gcval = *gcit;  
    System::Console::WriteLine("{0} {1}]", gcval->first, gcval->second);  
    return (0);  
}
```

```
[a 1] [b 2] [c 3]  
[a 1] [b 2] [c 3]  
[c 3]
```

## hash\_map::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

## Syntax

```
typedef GValue generic_value;
```

## Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

## Example

```

// cliext_hash_map_generic_value.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_map::generic_container^ gc1 = %c1;
    for each (Myhash_map::value_type elem in gc1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_map::generic_iterator gcit = gc1->begin();
    Myhash_map::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} {1}\n", gcval->first, gcval->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]

```

## hash\_map::hash\_delegate (STL/CLR)

Finds an element that matches a specified key.

### Syntax

```

hasher^ hash_delegate();

```

### Remarks

The member function returns the delegate used to convert a key value to an integer. You use it to hash a key.

### Example

```

// cliext_hash_map_hash_delegate.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    Myhash_map::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}

```

```

hash(L'a') = 1616896120
hash(L'b') = 570892832

```

## hash\_map::hash\_map (STL/CLR)

Constructs a container object.

### Syntax

```

hash_map();
explicit hash_map(key_compare^ pred);
hash_map(key_compare^ pred, hasher^ hashfn);
hash_map(hash_map<Key, Mapped>% right);
hash_map(hash_map<Key, Mapped>^ right);
template<typename InIter>
    hash_maphash_map(InIter first, InIter last);
template<typename InIter>
    hash_map(InIter first, InIter last,
        key_compare^ pred);
template<typename InIter>
    hash_map(InIter first, InIter last,
        key_compare^ pred, hasher^ hashfn);
hash_map(System::Collections::Generic::IEnumerable<GValue>^ right);
hash_map(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);
hash_map(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred, hasher^ hashfn);

```

### Parameters

*first*

Beginning of range to insert.

*hashfn*

Hash function for mapping keys to buckets.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

### Remarks

The constructor:

```
hash_map();
```

initializes the controlled sequence with no elements, with the default ordering predicate `key_compare()`, and with the default hash function. You use it to specify an empty initial controlled sequence, with the default ordering predicate and hash function.

The constructor:

```
explicit hash_map(key_compare^ pred);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*, and with the default hash function. You use it to specify an empty initial controlled sequence, with the specified ordering predicate and the default hash function.

The constructor:

```
hash_map(key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*, and with the hash function *hashfn*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate and hash function.

The constructor:

```
hash_map(hash_map<Key, Mapped>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`], with the default ordering predicate, and with the default hash function. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `hash_map` object *right*, with the default ordering predicate and hash function.

The constructor:

```
hash_map(hash_map<Key, Mapped>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`], with the default ordering predicate, and with the default hash function. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `hash_map` object *right*, with the default ordering predicate and hash function.

The constructor:

```
template<typename InIter> hash_map(InIter first, InIter last);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the default ordering predicate, and with the default hash function. You use it to make the controlled sequence a copy of another sequence, with the default ordering predicate and hash function.

The constructor:

```
template<typename InIter> hash_map(InIter first, InIter last, key_compare^ pred);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*, and with the default hash function. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate and the default hash function.

The constructor:

```
template<typename InIter> hash_map(InIter first, InIter last, key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*, and with

the hash function *hashfn*. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate and hash function.

The constructor:

```
hash_map(System::Collections::Generic::IEnumerable<Key>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the default ordering predicate, and with the default hash function. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the default ordering predicate and hash function.

The constructor:

```
hash_map(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*, and with the default hash function. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate and default hash function.

The constructor:

```
hash_map(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*, and with the hash function *hashfn*. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate and hash function.

## Example

```
// cliext_hash_map_construct.cpp
// compile with: /clr
#include <cliext/hash_map>

int myfun(wchar_t key)
    { // hash a key
    return (key ^ 0xdeadbeef);
    }

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
// construct an empty container
Myhash_map c1;
System::Console::WriteLine("size() = {0}", c1.size());

c1.insert(Myhash_map::make_value(L'a', 1));
c1.insert(Myhash_map::make_value(L'b', 2));
c1.insert(Myhash_map::make_value(L'c', 3));
for each (Myhash_map::value_type elem in c1)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an ordering rule
Myhash_map c2 = cliext::greater_equal<wchar_t>();
System::Console::WriteLine("size() = {0}", c2.size());

c2.insert(c1.begin(), c1.end());
for each (Myhash_map::value_type elem in c2)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an ordering rule and hash function
Myhash_map c2h(cliext::greater_equal<wchar_t>(),
    gcnew Myhash_map::hasher(&myfun));
System::Console::WriteLine("size() = {0}", c2h.size());
```

```

c2h.insert(c1.begin(), c1.end());
for each (Myhash_map::value_type elem in c2h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an iterator range
Myhash_map c3(c1.begin(), c1.end());
for each (Myhash_map::value_type elem in c3)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Myhash_map c4(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>());
for each (Myhash_map::value_type elem in c4)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule and hash function
Myhash_map c4h(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_map::hasher(&myfun));
for each (Myhash_map::value_type elem in c4h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an enumeration
Myhash_map c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_map::value_type>^)%c3);
for each (Myhash_map::value_type elem in c5)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myhash_map c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_map::value_type>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (Myhash_map::value_type elem in c6)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule and hash function
Myhash_map c6h( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_map::value_type>^)%c3,
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_map::hasher(&myfun));
for each (Myhash_map::value_type elem in c6h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct by copying another container
Myhash_map c7(c4);
for each (Myhash_map::value_type elem in c7)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying a container handle
Myhash_map c8(%c3);
for each (Myhash_map::value_type elem in c8)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
return (0);

```

```
}
```

```
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

## hash\_map::hasher (STL/CLR)

The hashing delegate for a key.

### Syntax

```
Microsoft::VisualC::StlClr::UnaryDelegate<GKey, int>
    hasher;
```

### Remarks

The type describes a delegate that converts a key value to an integer.

### Example

```
// cliext_hash_map_hasher.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    Myhash_map::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

## hash\_map::insert (STL/CLR)

Adds elements.

### Syntax

```
cliext::pair<iterator, bool> insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);
```

## Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Key value to insert.

*where*

Where in container to insert (hint only).

## Remarks

Each of the member functions inserts a sequence specified by the remaining operands.

The first member function endeavors to insert an element with value `val`, and returns a pair of values `x`. If `x.second` is true, `x.first` designates the newly inserted element; otherwise `x.first` designates an element with equivalent ordering that already exists and no new element is inserted. You use it to insert a single element.

The second member function inserts an element with value `val`, using `where` as a hint (to improve performance), and returns an iterator that designates the newly inserted element. You use it to insert a single element which might be adjacent to an element you know.

The third member function inserts the sequence [`first`, `last`). You use it to insert zero or more elements copied from another sequence.

The fourth member function inserts the sequence designated by the `right`. You use it to insert a sequence described by an enumerator.

Each element insertion takes time proportional to the logarithm of the number of elements in the controlled sequence. Insertion can occur in amortized constant time, however, given a hint that designates an element adjacent to the insertion point.

## Example

```

// cliext_hash_map_insert.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
typedef Myhash_map::pair_iter_bool Pairib;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Pairib pair1 =
        c1.insert(Myhash_map::make_value(L'x', 24));
    System::Console::WriteLine("insert([L'x' 24]) = [{0} {1}] {2}",
        pair1.first->first, pair1.first->second, pair1.second);

    pair1 = c1.insert(Myhash_map::make_value(L'b', 2));
    System::Console::WriteLine("insert([L'b' 2]) = [{0} {1}] {2}",
        pair1.first->first, pair1.first->second, pair1.second);

    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value with hint
    Myhash_map::iterator it =
        c1.insert(c1.begin(), Myhash_map::make_value(L'y', 25));
    System::Console::WriteLine("insert(begin(), [L'y' 25]) = [{0} {1}]",
        it->first, it->second);
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an iterator range
    Myhash_map c2;
    it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (Myhash_map::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an enumeration
    Myhash_map c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::
        IEnumerable<Myhash_map::value_type>^)%c1);
    for each (Myhash_map::value_type elem in c3)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
insert([L'x' 24]) = [x 24] True
insert([L'b' 2]) = [b 2] False
[a 1] [b 2] [c 3] [x 24]
insert(begin(), [L'y' 25]) = [y 25]
[a 1] [b 2] [c 3] [x 24] [y 25]
[a 1] [b 2] [c 3] [x 24]
[a 1] [b 2] [c 3] [x 24] [y 25]
```

## hash\_map::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

### Remarks

The type describes an object of unspecified type `T1` that can serve as a bidirectional iterator for the controlled sequence.

### Example

```
// cliext_hash_map_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_map::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("[{0} {1}] ", it->first, it->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

## hash\_map::key\_comp (STL/CLR)

Copies the ordering delegate for two keys.

### Syntax

```
key_compare^key_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two keys.

## Example

```
// cliext_hash_map_key_comp.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    Myhash_map::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_map c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}
```

```
compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True
```

## hash\_map::key\_compare (STL/CLR)

The ordering delegate for two keys.

### Syntax

```
Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;
```

### Remarks

The type is a synonym for the delegate that determines the ordering of its key arguments.

## Example

```

// cliext_hash_map_key_compare.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    Myhash_map::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_map c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## hash\_map::key\_type (STL/CLR)

The type of an ordering key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter *Key*.

### Example

```

// cliext_hash_map_key_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using key_type
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myhash_map::key_type val = it->first;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## hash\_map::load\_factor (STL/CLR)

Counts the average elements per bucket.

### Syntax

```
float load_factor();
```

### Remarks

The member function returns `(float) hash_map::size (STL/CLR) () / hash_map::bucket_count (STL/CLR) ()`.

You use it to determine the average bucket size.

### Example

```

// cliext_hash_map_load_factor.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1 = gcnew Myhash_map;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_map::lower\_bound (STL/CLR)

Finds beginning of range that matches a specified key.

### Syntax

```
iterator lower_bound(key_type key);
```

## Parameters

*key*

Key value to search for.

## Remarks

The member function determines the first element *x* in the controlled sequence that hashes to the same bucket as *key* and has equivalent ordering to *key*. If no such element exists, it returns `hash_map::end (STL/CLR)()`; otherwise it returns an iterator that designates *x*. You use it to locate the beginning of a sequence of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_hash_map_lower_bound.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    Myhash_map::iterator it = c1.lower_bound(L'a');
    System::Console::WriteLine("*lower_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.lower_bound(L'b');
    System::Console::WriteLine("*lower_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
lower_bound(L'x') == end() = True
*lower_bound(L'a') = [a 1]
*lower_bound(L'b') = [b 2]
```

## hash\_map::make\_value (STL/CLR)

Constructs a value object.

## Syntax

```
static value_type make_value(key_type key, mapped_type mapped);
```

## Parameters

*key*

Key value to use.

*mapped*

Mapped value to search for.

## Remarks

The member function returns a `value_type` object whose key is *key* and whose mapped value is *mapped*. You use it to compose an object suitable for use with several other member functions.

## Example

```
// cliext_hash_map_make_value.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

## hash\_map::mapped\_type (STL/CLR)

The type of a mapped value associated with each key.

## Syntax

```
typedef Mapped mapped_type;
```

## Remarks

The type is a synonym for the template parameter `Mapped`.

## Example

```

// cliext_hash_map_mapped_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using mapped_type
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in mapped_type object
        Myhash_map::mapped_type val = it->second;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

1 2 3

## hash\_map::max\_load\_factor (STL/CLR)

Gets or sets the maximum elements per bucket.

### Syntax

```

float max_load_factor();
void max_load_factor(float new_factor);

```

### Parameters

*new\_factor*

New maximum load factor to store.

### Remarks

The first member function returns the current stored maximum load factor. You use it to determine the maximum average bucket size.

The second member function replaces the store maximum load factor with *new\_factor*. No automatic rehashing occurs until a subsequent insert.

### Example

```

// cliext_hash_map_max_load_factor.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1 = gcnew Myhash_map;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_map::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
hash_map<Key, Mapped>% operator=(hash_map<Key, Mapped>% right);
```

## Parameters

*right*

Container to copy.

## Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

## Example

```
// cliext_hash_map_operator_as.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Myhash_map c2;
    c2 = c1;
    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

## hash\_map::operator(STL/CLR)

Maps a key to its associated mapped value.

## Syntax

```
mapped_type operator[](key_type key);
```

## Parameters

*key*

Key value to search for.

## Remarks

The member functions endeavors to find an element with equivalent ordering to *key*. If it finds one, it returns the associated mapped value; otherwise, it inserts `value_type(key, mapped_type())` and returns the associated (default) mapped value. You use it to look up a mapped value given its associated key, or to ensure that an entry exists for the key if none is found.

## Example

```

// cliext_hash_map_operator_sub.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("c1[{0}] = {1}",
        L'A', c1[L'A']);
    System::Console::WriteLine("c1[{0}] = {1}",
        L'b', c1[L'b']);

    // redisplay altered contents
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // alter mapped values and redisplay
    c1[L'A'] = 10;
    c1[L'c'] = 13;
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
c1[A] = 0
c1[b] = 2
[a 1] [A 0] [b 2] [c 3]
[a 1] [A 10] [b 2] [c 13]

```

## hash\_map::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

### Syntax

```
reverse_iterator rbegin();
```

### Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the **beginning** of the reverse sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_hash_map_rbegin.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_map::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("++rbegin() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*rbegin() = [c 3]
*++rbegin() = [b 2]

```

## hash\_map::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_hash_map_reference.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_map::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        Myhash_map::reference ref = *it;
        System::Console::Write("[{0} {1}] ", ref->first, ref->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## hash\_map::rehash (STL/CLR)

Rebuilds the hash table.

### Syntax

```
void rehash();
```

### Remarks

The member function rebuilds the hash table, ensuring that [hash\\_map::load\\_factor \(STL/CLR\)](#) ()  $\leq$  [hash\\_map::max\\_load\\_factor \(STL/CLR\)](#). Otherwise, the hash table increases in size only as needed after an insertion. (It never automatically decreases in size.) You use it to adjust the size of the hash table.

### Example

```

// cliext_hash_map_rehash.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1 = gcnew Myhash_map;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_map::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

## Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

## Example

```
// cliext_hash_map_rend.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_map::reverse_iterator rit = c1.rend();
    --rit;
    --rit;
    System::Console::WriteLine(" --- --rend() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine(" --- rend() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*-- --rend() = [b 2]
*--rend() = [a 1]
```

## hash\_map::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

## Syntax

```
typedef T3 reverse_iterator;
```

## Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

## Example

```
// cliext_hash_map_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Myhash_map::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("[{0} {1}] ", rit->first, rit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

## hash\_map::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [hash\\_map::empty \(STL/CLR\) \(\)](#).

### Example

```

// cliext_hash_map_size.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(Myhash_map::make_value(L'd', 4));
    c1.insert(Myhash_map::make_value(L'e', 5));
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0 after clearing
size() = 2 after adding 2

```

## hash\_map::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_hash_map_size_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_map::size_type diff = 0;
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3

```

## hash\_map::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```

void swap(hash_map<Key, Mapped>% right);

```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_hash_map_swap.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Myhash_map c2;
    c2.insert(Myhash_map::make_value(L'd', 4));
    c2.insert(Myhash_map::make_value(L'e', 5));
    c2.insert(Myhash_map::make_value(L'f', 6));
    for each (Myhash_map::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    for each (Myhash_map::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[d 4] [e 5] [f 6]
[d 4] [e 5] [f 6]
[a 1] [b 2] [c 3]

```

## hash\_map::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<value_type>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_hash_map_to_array.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // copy the container and modify it
    cli::array<Myhash_map::value_type>^ a1 = c1.to_array();

    c1.insert(Myhash_map::make_value(L'd', 4));
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (Myhash_map::value_type elem in a1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3]

```

## hash\_map::upper\_bound (STL/CLR)

Finds end of range that matches a specified key.

### Syntax

```

iterator upper_bound(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the last element *x* in the controlled sequence that hashes to the same bucket as *key* and has equivalent ordering to *key*. If no such element exists, or if *x* is the last element in the controlled sequence, it returns [hash\\_map::end \(STL/CLR\) \(\)](#); otherwise it returns an iterator that designates the first element beyond *x*. You use it to locate the end of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_hash_map_upper_bound.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    Myhash_map::iterator it = c1.upper_bound(L'a');
    System::Console::WriteLine("*upper_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.upper_bound(L'b');
    System::Console::WriteLine("*upper_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
upper_bound(L'x') == end() = True
*upper_bound(L'a') = [b 2]
*upper_bound(L'b') = [c 3]

```

## hash\_map::value\_comp (STL/CLR)

Copies the ordering delegate for two element values.

### Syntax

```
value_compare^ value_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two element values.

### Example

```

// cliext_hash_map_value_comp.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    Myhash_map::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Myhash_map::make_value(L'a', 1),
            Myhash_map::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Myhash_map::make_value(L'a', 1),
            Myhash_map::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Myhash_map::make_value(L'b', 2),
            Myhash_map::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}

```

```

compare([L'a', 1], [L'a', 1]) = True
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False

```

## hash\_map::value\_compare (STL/CLR)

The ordering delegate for two element values.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
value_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its value arguments.

### Example

```

// cliext_hash_map_value_compare.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    Myhash_map::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Myhash_map::make_value(L'a', 1),
            Myhash_map::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Myhash_map::make_value(L'a', 1),
            Myhash_map::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Myhash_map::make_value(L'b', 2),
            Myhash_map::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}

```

```

compare([L'a', 1], [L'a', 1]) = True
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False

```

## hash\_map::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef generic_value value_type;
```

### Remarks

The type is a synonym for `generic_value`.

### Example

```
// cliext_hash_map_value_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using value_type
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Myhash_map::value_type val = *it;
        System::Console::Write("[{0} {1}] ", val->first, val->second);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

# hash\_multimap (STL/CLR)

9/20/2022 • 43 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `hash_multimap` to manage a sequence of elements as a hash table, each table entry storing a bidirectional linked list of nodes, and each node storing one element. An element consists of a key, for ordering the sequence, and a mapped value, which goes along for the ride.

In the description below, `GValue` is the same as:

`Microsoft::VisualC::StlClr::GenericPair<GKey, GMapped>`

where:

`GKey` is the same as *Key* unless the latter is a ref type, in which case it is `Key^`

`GMapped` is the same as *Mapped* unless the latter is a ref type, in which case it is `Mapped^`

## Syntax

```
template<typename Key,
         typename Mapped>
ref class hash_multimap
    : public
        System::ICloneable,
        System::Collections::IEnumerable,
        System::Collections::ICollection,
        System::Collections::Generic::IEnumerable<GValue>,
        System::Collections::Generic::ICollection<GValue>,
        System::Collections::Generic::IList<GValue>,
        Microsoft::VisualC::StlClr::IHash<Gkey, GValue>
{ .... };
```

## Parameters

### *Key*

The type of the key component of an element in the controlled sequence.

### *Mapped*

The type of the additional component of an element in the controlled sequence.

## Requirements

**Header:** <cliext/hash\_map>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<code>hash_multimap::const_iterator (STL/CLR)</code>	The type of a constant iterator for the controlled sequence.
<code>hash_multimap::const_reference (STL/CLR)</code>	The type of a constant reference to an element.

TYPE DEFINITION	DESCRIPTION
<code>hash_multimap::const_reverse_iterator (STL/CLR)</code>	The type of a constant reverse iterator for the controlled sequence.
<code>hash_multimap::difference_type (STL/CLR)</code>	The type of a (possibly signed) distance between two elements.
<code>hash_multimap::generic_container (STL/CLR)</code>	The type of the generic interface for the container.
<code>hash_multimap::generic_iterator (STL/CLR)</code>	The type of an iterator for the generic interface for the container.
<code>hash_multimap::generic_reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the generic interface for the container.
<code>hash_multimap::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container.
<code>hash_multimap::hasher (STL/CLR)</code>	The hashing delegate for a key.
<code>hash_multimap::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>hash_multimap::key_compare (STL/CLR)</code>	The ordering delegate for two keys.
<code>hash_multimap::key_type (STL/CLR)</code>	The type of an ordering key.
<code>hash_multimap::mapped_type (STL/CLR)</code>	The type of the mapped value associated with each key.
<code>hash_multimap::reference (STL/CLR)</code>	The type of a reference to an element.
<code>hash_multimap::reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the controlled sequence.
<code>hash_multimap::size_type (STL/CLR)</code>	The type of a (non-negative) distance between two elements.
<code>hash_multimap::value_compare (STL/CLR)</code>	The ordering delegate for two element values.
<code>hash_multimap::value_type (STL/CLR)</code>	The type of an element.
MEMBER FUNCTION	DESCRIPTION
<code>hash_multimap::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>hash_multimap::bucket_count (STL/CLR)</code>	Counts the number of buckets.
<code>hash_multimap::clear (STL/CLR)</code>	Removes all elements.
<code>hash_multimap::count (STL/CLR)</code>	Counts elements matching a specified key.
<code>hash_multimap::empty (STL/CLR)</code>	Tests whether no elements are present.
<code>hash_multimap::end (STL/CLR)</code>	Designates the end of the controlled sequence.

MEMBER FUNCTION	DESCRIPTION
<code>hash_multimap::equal_range (STL/CLR)</code>	Finds range that matches a specified key.
<code>hash_multimap::erase (STL/CLR)</code>	Removes elements at specified positions.
<code>hash_multimap::find (STL/CLR)</code>	Finds an element that matches a specified key.
<code>hash_multimap::hash_delegate (STL/CLR)</code>	Copies the hashing delegate for a key.
<code>hash_multimap::hash_multimap (STL/CLR)</code>	Constructs a container object.
<code>hash_multimap::insert (STL/CLR)</code>	Adds elements.
<code>hash_multimap::key_comp (STL/CLR)</code>	Copies the ordering delegate for two keys.
<code>hash_multimap::load_factor (STL/CLR)</code>	Counts the average elements per bucket.
<code>hash_multimap::lower_bound (STL/CLR)</code>	Finds beginning of range that matches a specified key.
<code>hash_multimap::make_value (STL/CLR)</code>	Constructs a value object.
<code>hash_multimap::max_load_factor (STL/CLR)</code>	Gets or sets the maximum elements per bucket.
<code>hash_multimap::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.
<code>hash_multimap::rehash (STL/CLR)</code>	Rebuilds the hash table.
<code>hash_multimap::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>hash_multimap::size (STL/CLR)</code>	Counts the number of elements.
<code>hash_multimap::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>hash_multimap::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>hash_multimap::upper_bound (STL/CLR)</code>	Finds end of range that matches a specified key.
<code>hash_multimap::value_comp (STL/CLR)</code>	Copies the ordering delegate for two element values.
OPERATOR	DESCRIPTION
<code>hash_multimap::operator= (STL/CLR)</code>	Replaces the controlled sequence.

## Interfaces

INTERFACE	DESCRIPTION
<code>ICloneable</code>	Duplicate an object.

INTERFACE	DESCRIPTION
IEnumerable	Sequence through elements.
ICollection	Maintain group of elements.
IEnumerable<T>	Sequence through typed elements.
ICollection<T>	Maintain group of typed elements.
IHash<Key, Value>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes in a bidirectional linked list. To speed access, the object also maintains a varying-length array of pointers into the list (the hash table), effectively managing the whole list as a sequence of sublists, or buckets. It inserts elements into a bucket that it keeps ordered by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements.

The object orders each bucket it controls by calling a stored delegate object of type [hash\\_set::key\\_compare \(STL/CLR\)](#). You can specify the stored delegate object when you construct the `hash_set`; if you specify no delegate object, the default is the comparison `operator<=(key_type, key_type)`.

You access the stored delegate object by calling the member function `hash_set::key_comp (STL/CLR)()`. Such a delegate object must define equivalent ordering between keys of type [hash\\_set::key\\_type \(STL/CLR\)](#). That means, for any two keys `x` and `y`:

`key_comp()(x, y)` returns the same Boolean result on every call.

If `key_comp()(x, y) && key_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

Any ordering rule that behaves like `operator<=(key_type, key_type)`, `operator>=(key_type, key_type)` or `operator==(key_type, key_type)` defines equivalent ordering.

Note that the container ensures only that elements whose keys have equivalent ordering (and which hash to the same integer value) are adjacent within a bucket. Unlike template class [hash\\_map \(STL/CLR\)](#), an object of template class `hash_multimap` does not require that keys for all elements are unique. (Two or more keys can have equivalent ordering.)

The object determines which bucket should contain a given ordering key by calling a stored delegate object of type [hash\\_set::hasher \(STL/CLR\)](#). You access this stored object by calling the member function `hash_set::hash_delegate (STL/CLR)()` to obtain an integer value that depends on the key value. You can specify the stored delegate object when you construct the `hash_set`; if you specify no delegate object, the default is the function `System::Object::hash_value(key_type)`. That means, for any keys `x` and `y`:

`hash_delegate()(x)` returns the same integer result on every call.

If `x` and `y` have equivalent ordering, then `hash_delegate()(x)` should return the same integer result as `hash_delegate()(y)`.

Each element contains a separate key and a mapped value. The sequence is represented in a way that permits lookup, insertion, and removal of an arbitrary element with a number of operations that is independent of the number of elements in the sequence (constant time) -- at least in the best of cases. Moreover, inserting an element invalidates no iterators, and removing an element invalidates only those iterators which point at the

removed element.

If hashed values are not uniformly distributed, however, a hash table can degenerate. In the extreme -- for a hash function that always returns the same value -- lookup, insertion, and removal are proportional to the number of elements in the sequence (linear time). The container endeavors to choose a reasonable hash function, mean bucket size, and hash-table size (total number of buckets), but you can override any or all of these choices. See, for example, the functions [hash\\_set::max\\_load\\_factor \(STL/CLR\)](#) and [hash\\_set::rehash \(STL/CLR\)](#).

A hash\_multimap supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by [hash\\_multimap::end \(STL/CLR\)](#). You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a hash\_multimap iterator to reach the head node, and it will then compare equal to [end\(\)](#). But you cannot dereference the iterator returned by [end\(\)](#).

Note that you cannot refer to a hash\_multimap element directly given its numerical position -- that requires a random-access iterator.

A hash\_multimap iterator stores a handle to its associated hash\_multimap node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A hash\_multimap iterator remains valid so long as its associated hash\_multimap node is associated with some hash\_multimap. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to [end\(\)](#).

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### hash\_multimap::begin (STL/CLR)

Designates the beginning of the controlled sequence.

#### Syntax

```
iterator begin();
```

#### Remarks

The member function returns a bidirectional iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the [current](#) beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

#### Example

```

// cliext_hash_multimap_begin.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_multimap::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = [{0} {1}]\n",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*++begin() = [{0} {1}]\n",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*begin() = [a 1]
*++begin() = [b 2]

```

## hash\_multimap::bucket\_count (STL/CLR)

Counts the number of buckets.

### Syntax

```
int bucket_count();
```

### Remarks

The member functions returns the current number of buckets. You use it to determine the size of the hash table.

### Example

```

// cliext_hash_multimap_bucket_count.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1 = gcnew Myhash_multimap;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_multimap::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

## Remarks

The member function effectively calls `hash_multimap::erase (STL/CLR) (` `hash_multimap::begin (STL/CLR) ()`, `hash_multimap::end (STL/CLR) ()`). You use it to ensure that the controlled sequence is empty.

## Example

```
// cliext_hash_multimap_clear.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));

    // display contents " [a 1] [b 2]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2]
size() = 0
```

## hash\_multimap::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

## Syntax

```
typedef T2 const_iterator;
```

## Remarks

The type describes an object of unspecified type `T2` that can serve as a constant bidirectional iterator for the controlled sequence.

## Example

```

// cliext_hash_multimap_const_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_multimap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("[{0} {1}] ", cit->first, cit->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## hash\_multimap::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_hash_multimap_const_reference.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_multimap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Myhash_multimap::const_reference cref = *cit;
        System::Console::Write("[{0} {1}] ", cref->first, cref->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
```

## hash\_multimap::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```
// cliext_hash_multimap_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Myhash_multimap::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("[{0} {1}] ", crit->first, crit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

## hash\_multimap::count (STL/CLR)

Finds the number of elements matching a specified key.

### Syntax

```
size_type count(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns the number of elements in the controlled sequence that have equivalent ordering with *key*. You use it to determine the number of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_hash_multimap_count.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

## hash\_multimap::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a possibly negative element count.

## Example

```

// cliext_hash_multimap_difference_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_multimap::difference_type diff = 0;
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Myhash_multimap::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3
begin()-end() = -3

```

## hash\_multimap::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

### Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [hash\\_multimap::size \(STL/CLR\)](#) `() == 0`. You use it to test whether the hash\_multimap is empty.

### Example

```

// cliext_hash_multimap_empty.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 3
empty() = False
size() = 0
empty() = True

```

## hash\_multimap::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a bidirectional iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_hash_multimap_end.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect last two items
    Myhash_multimap::iterator it = c1.end();
    --it;
    --it;
    System::Console::WriteLine("---- --end() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("----end() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
---- --end() = [b 2]
----end() = [c 3]

```

## hash\_multimap::equal\_range (STL/CLR)

Finds range that matches a specified key.

### Syntax

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns a pair of iterators `cliext::pair<iterator, iterator>(hash_multimap::lower_bound (STL/CLR) (key), hash_multimap::upper_bound (STL/CLR) (key))`. You use it to determine the range of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_hash_multimap_equal_range.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
typedef Myhash_multimap::pair_iter_iter Pairii;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("[{0} {1}] ",
            pair1.first->first, pair1.first->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
equal_range(L'x') empty = True
[b 2]

```

## hash\_multimap::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

### Parameters

*first*

Beginning of range to erase.

*key*

Key value to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*, and returns an iterator that designates the first element remaining beyond the element removed, or `hash_multimap::end` ([\(STL/CLR\)](#)) if no such element exists. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [`first`, `last`), and returns an iterator that designates the first element remaining beyond any elements removed, or `end()` if no such element exists.. You use it to remove zero or more contiguous elements.

The third member function removes any element of the controlled sequence whose key has equivalent ordering to *key*, and returns a count of the number of elements removed. You use it to remove and count all elements that match a specified key.

Each element erasure takes time proportional to the logarithm of the number of elements in the controlled sequence.

## Example

```
// cliext_hash_multimap_erase.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    cliext::hash_multimap<wchar_t, int> c1;
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'a', 1));
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'b', 2));
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (cliext::hash_multimap<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase an element and reinspect
    cliext::hash_multimap<wchar_t, int>::iterator it =
        c1.erase(c1.begin());
    System::Console::WriteLine("erase(begin()) = [{0} {1}]",
        it->first, it->second);

    // add elements and display " b c d e"
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'd', 4));
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'e', 5));
    for each (cliext::hash_multimap<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase all but end
    it = c1.end();
    it = c1.erase(c1.begin(), --it);
    System::Console::WriteLine("erase(begin(), end()-1) = [{0} {1}]",
        it->first, it->second);
    System::Console::WriteLine("size() = {0}", c1.size());

    // erase end
    System::Console::WriteLine("erase(L'x') = {0}", c1.erase(L'x'));
    System::Console::WriteLine("erase(L'e') = {0}", c1.erase(L'e'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
erase(begin()) = [b 2]
[b 2] [c 3] [d 4] [e 5]
erase(begin(), end()-1) = [e 5]
size() = 1
erase(L'x') = 0
erase(L'e') = 1
```

## hash\_multimap::find (STL/CLR)

Finds an element that matches a specified key.

### Syntax

```
iterator find(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

If at least one element in the controlled sequence has equivalent ordering with *key*, the member function returns an iterator designating one of those elements; otherwise it returns [hash\\_multimap::end \(STL/CLR\)](#) (). You use it to locate an element currently in the controlled sequence that matches a specified key.

### Example

```
// cliext_hash_multimap_find.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());

    Myhash_multimap::iterator it = c1.find(L'b');
    System::Console::WriteLine("find {0} = [{1} {2}]",
        L'b', it->first, it->second);

    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
find A = False
find b = [b 2]
find C = False
```

## hash\_multimap::generic\_container (STL/CLR)

The type of the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::
    IHash<GKey, GValue>
generic_container;
```

### Remarks

The type describes the generic interface for this template container class.

### Example

```
// cliext_hash_multimap_generic_container.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multimap::generic_container^ gc1 = %c1;
    for each (Myhash_multimap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(Myhash_multimap::make_value(L'd', 4));
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(Myhash_multimap::make_value(L'e', 5));
    for each (Myhash_multimap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3] [d 4] [e 5]
```

## hash\_multimap::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

### Example

```
// cliext_hash_multimap_generic_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multimap::generic_container^ gc1 = %c1;
    for each (Myhash_multimap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_multimap::generic_iterator gcit = gc1->begin();
    Myhash_multimap::generic_value gcval = *gcit;
    System::Console::Write("[{0} {1}] ", gcval->first, gcval->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]
```

## hash\_multimap::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::  
    ReverseRandomAccessIterator<generic_value>  
    generic_reverse_iterator;
```

## Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

## Example

```
// cliext_hash_multimap_generic_reverse_iterator.cpp  
// compile with: /clr  
#include <cliext/hash_map>  
  
typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;  
int main()  
{  
    Myhash_multimap c1;  
    c1.insert(Myhash_multimap::make_value(L'a', 1));  
    c1.insert(Myhash_multimap::make_value(L'b', 2));  
    c1.insert(Myhash_multimap::make_value(L'c', 3));  
  
    // display contents " [a 1] [b 2] [c 3]"  
    for each (Myhash_multimap::value_type elem in c1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // construct a generic container  
    Myhash_multimap::generic_container^ gc1 = %c1;  
    for each (Myhash_multimap::value_type elem in gc1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // get an element and display it  
    Myhash_multimap::generic_reverse_iterator gcit = gc1->rbegin();  
    Myhash_multimap::generic_value gcval = *gcit;  
    System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);  
    return (0);  
}
```

```
[a 1] [b 2] [c 3]  
[a 1] [b 2] [c 3]  
[c 3]
```

## hash\_multimap::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

## Syntax

```
typedef GValue generic_value;
```

## Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

## Example

```

// cliext_hash_multimap_generic_value.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multimap::generic_container^ gc1 = %c1;
    for each (Myhash_multimap::value_type elem in gc1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_multimap::generic_iterator gcit = gc1->begin();
    Myhash_multimap::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} {1}\n", gcval->first, gcval->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]

```

## hash\_multimap::hash\_delegate (STL/CLR)

Finds an element that matches a specified key.

### Syntax

```

hasher^ hash_delegate();

```

### Remarks

The member function returns the delegate used to convert a key value to an integer. You use it to hash a key.

### Example

```

// cliext_hash_multimap_hash_delegate.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}

```

```

hash(L'a') = 1616896120
hash(L'b') = 570892832

```

## hash\_multimap::hash\_multimap (STL/CLR)

Constructs a container object.

### Syntax

```

hash_multimap();
explicit hash_multimap(key_compare^ pred);
hash_multimap(key_compare^ pred, hasher^ hashfn);
hash_multimap(hash_multimap<Key, Mapped>% right);
hash_multimap(hash_multimap<Key, Mapped>^ right);
template<typename InIter>
    hash_multimap(hash_multimap<InIter first, InIter last>,
template<typename InIter>
    hash_multimap<InIter first, InIter last,
        key_compare^ pred>;
template<typename InIter>
    hash_multimap<InIter first, InIter last,
        key_compare^ pred, hasher^ hashfn>;
hash_multimap(System::Collections::Generic::IEnumerable<GValue>^ right);
hash_multimap(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);
hash_multimap(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred, hasher^ hashfn);

```

### Parameters

*first*

Beginning of range to insert.

*hashfn*

Hash function for mapping keys to buckets.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

### Remarks

The constructor:

```
hash_multimap();
```

initializes the controlled sequence with no elements, with the default ordering predicate `key_compare()`, and with the default hash function. You use it to specify an empty initial controlled sequence, with the default ordering predicate and hash function.

The constructor:

```
explicit hash_multimap(key_compare^ pred);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*, and with the default hash function. You use it to specify an empty initial controlled sequence, with the specified ordering predicate and the default hash function.

The constructor:

```
hash_multimap(key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*, and with the hash function *hashfn*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate and hash function.

The constructor:

```
hash_multimap(hash_multimap<Key, Mapped>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`], with the default ordering predicate, and with the default hash function. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `hash_multimap` object *right*, with the default ordering predicate and hash function.

The constructor:

```
hash_multimap(hash_multimap<Key, Mapped>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`], with the default ordering predicate, and with the default hash function. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `hash_multimap` object *right*, with the default ordering predicate and hash function.

The constructor:

```
template<typename InIter> hash_multimap(InIter first, InIter last);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the default ordering predicate, and with the default hash function. You use it to make the controlled sequence a copy of another sequence, with the default ordering predicate and hash function.

The constructor:

```
template<typename InIter> hash_multimap(InIter first, InIter last, key_compare^ pred);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*, and with the default hash function. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate and the default hash function.

The constructor:

```
template<typename InIter> hash_multimap(InIter first, InIter last, key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with the sequence [`first`, `last`), with the ordering predicate `pred`, and with the hash function `hashfn`. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate and hash function.

The constructor:

```
hash_multimap(System::Collections::Generic::IEnumerable<Key>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator `right`, with the default ordering predicate, and with the default hash function. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the default ordering predicate and hash function.

The constructor:

```
hash_multimap(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

initializes the controlled sequence with the sequence designated by the enumerator `right`, with the ordering predicate `pred`, and with the default hash function. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate and default hash function.

The constructor:

```
hash_multimap(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with the sequence designated by the enumerator `right`, with the ordering predicate `pred`, and with the hash function `hashfn`. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate and hash function.

## Example

```
// cliext_hash_multimap_construct.cpp
// compile with: /clr
#include <cliext/hash_map>

int myfun(wchar_t key)
{ // hash a key
    return (key ^ 0xdeadbeef);
}

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
// construct an empty container
    Myhash_multimap c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

// construct with an ordering rule
    Myhash_multimap c2 = cliext::greater_equal<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    c2.insert(c1.begin(), c1.end());
    for each (Myhash_multimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

// construct with an ordering rule and hash function
    Myhash_multimap c2h(cliext::greater_equal<wchar_t>(),
        gcnew Myhash_multimap::hasher(&myfun));
```

```

System::Console::WriteLine("size() = {0}", c2h.size());

c2h.insert(c1.begin(), c1.end());
for each (Myhash_multimap::value_type elem in c2h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an iterator range
Myhash_multimap c3(c1.begin(), c1.end());
for each (Myhash_multimap::value_type elem in c3)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Myhash_multimap c4(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>());
for each (Myhash_multimap::value_type elem in c4)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule and hash function
Myhash_multimap c4h(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_multimap::hasher(&myfun));
for each (Myhash_multimap::value_type elem in c4h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an enumeration
Myhash_multimap c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_multimap::value_type>^)%c3);
for each (Myhash_multimap::value_type elem in c5)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myhash_multimap c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_multimap::value_type>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (Myhash_multimap::value_type elem in c6)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule and hash function
Myhash_multimap c6h( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_multimap::value_type>^)%c3,
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_multimap::hasher(&myfun));
for each (Myhash_multimap::value_type elem in c6h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct by copying another container
Myhash_multimap c7(c4);
for each (Myhash_multimap::value_type elem in c7)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying a container handle
Myhash_multimap c8(%c3);
for each (Myhash_multimap::value_type elem in c8)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

```

```
System::Console::WriteLine(),
return (0);
}
```

```
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

## hash\_multimap::hasher (STL/CLR)

The hashing delegate for a key.

### Syntax

```
Microsoft::VisualC::StlClr::UnaryDelegate<GKey, int>
hasher;
```

### Remarks

The type describes a delegate that converts a key value to an integer.

### Example

```
// cliext_hash_multimap_hasher.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

## hash\_multimap::insert (STL/CLR)

Adds elements.

### Syntax

```
iterator insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumarable<value_type>^ right);
```

## Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Key value to insert.

*where*

Where in container to insert (hint only).

## Remarks

Each of the member functions inserts a sequence specified by the remaining operands.

The first member function inserts an element with value *val*, and returns an iterator that designates the newly inserted element. You use it to insert a single element.

The second member function inserts an element with value *val*, using *where* as a hint (to improve performance), and returns an iterator that designates the newly inserted element. You use it to insert a single element which might be adjacent to an element you know.

The third member function inserts the sequence [*first*, *last*). You use it to insert zero or more elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

Each element insertion takes time proportional to the logarithm of the number of elements in the controlled sequence. Insertion can occur in amortized constant time, however, given a hint that designates an element adjacent to the insertion point.

## Example

```

// cliext_hash_multimap_insert.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Myhash_multimap::iterator it =
        c1.insert(Myhash_multimap::make_value(L'x', 24));
    System::Console::WriteLine("insert([L'x' 24]) = [{0} {1}]",
        it->first, it->second);

    it = c1.insert(Myhash_multimap::make_value(L'b', 2));
    System::Console::WriteLine("insert([L'b' 2]) = [{0} {1}]",
        it->first, it->second);

    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value with hint
    it = c1.insert(c1.begin(), Myhash_multimap::make_value(L'y', 25));
    System::Console::WriteLine("insert(begin(), [L'y' 25]) = [{0} {1}]",
        it->first, it->second);
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an iterator range
    Myhash_multimap c2;
    it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (Myhash_multimap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an enumeration
    Myhash_multimap c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::
        IEnumerable<Myhash_multimap::value_type>)c1);
    for each (Myhash_multimap::value_type elem in c3)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
insert([L'x' 24]) = [x 24]
insert([L'b' 2]) = [b 2]
[a 1] [b 2] [b 2] [c 3] [x 24]
insert(begin(), [L'y' 25]) = [y 25]
[a 1] [b 2] [b 2] [c 3] [x 24] [y 25]
[a 1] [b 2] [b 2] [c 3] [x 24]
[a 1] [b 2] [b 2] [c 3] [x 24] [y 25]
```

## hash\_multimap::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

### Remarks

The type describes an object of unspecified type `T1` that can serve as a bidirectional iterator for the controlled sequence.

### Example

```
// cliext_hash_multimap_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_multimap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("[{0} {1}] ", it->first, it->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

## hash\_multimap::key\_comp (STL/CLR)

Copies the ordering delegate for two keys.

### Syntax

```
key_compare^key_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two keys.

## Example

```
// cliext_hash_multimap_key_comp.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_multimap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}
```

```
compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True
```

## hash\_multimap::key\_compare (STL/CLR)

The ordering delegate for two keys.

### Syntax

```
Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;
```

### Remarks

The type is a synonym for the delegate that determines the ordering of its key arguments.

## Example

```

// cliext_hash_multimap_key_compare.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_multimap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## hash\_multimap::key\_type (STL/CLR)

The type of an ordering key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter *Key*.

### Example

```

// cliext_hash_multimap_key_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using key_type
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myhash_multimap::key_type val = it->first;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## hash\_multimap::load\_factor (STL/CLR)

Counts the average elements per bucket.

### Syntax

```
float load_factor();
```

### Remarks

The member function returns `(float) hash_multimap::size (STL/CLR) () / hash_multimap::bucket_count (STL/CLR) ()`. You use it to determine the average bucket size.

### Example

```

// cliext_hash_multimap_load_factor.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1 = gcnew Myhash_multimap;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_multimap::lower\_bound (STL/CLR)

Finds beginning of range that matches a specified key.

### Syntax

```
iterator lower_bound(key_type key);
```

## Parameters

*key*

Key value to search for.

## Remarks

The member function determines the first element *x* in the controlled sequence that hashes to the same bucket as *key* and has equivalent ordering to *key*. If no such element exists, it returns `hash_multimap::end (STL/CLR) ()`; otherwise it returns an iterator that designates *x*. You use it to locate the beginning of a sequence of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_hash_multimap_lower_bound.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    Myhash_multimap::iterator it = c1.lower_bound(L'a');
    System::Console::WriteLine("*lower_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.lower_bound(L'b');
    System::Console::WriteLine("*lower_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
lower_bound(L'x') == end() = True
*lower_bound(L'a') = [a 1]
*lower_bound(L'b') = [b 2]
```

## hash\_multimap::make\_value (STL/CLR)

Constructs a value object.

## Syntax

```
static value_type make_value(key_type key, mapped_type mapped);
```

## Parameters

*key*

Key value to use.

*mapped*

Mapped value to search for.

### Remarks

The member function returns a `value_type` object whose key is *key* and whose mapped value is *mapped*. You use it to compose an object suitable for use with several other member functions.

### Example

```
// cliext_hash_multimap_make_value.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

## hash\_multimap::mapped\_type (STL/CLR)

The type of a mapped value associated with each key.

### Syntax

```
typedef Mapped mapped_type;
```

### Remarks

The type is a synonym for the template parameter *Mapped*.

### Example

```

// cliext_hash_multimap_mapped_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using mapped_type
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in mapped_type object
        Myhash_multimap::mapped_type val = it->second;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

1 2 3

## hash\_multimap::max\_load\_factor (STL/CLR)

Gets or sets the maximum elements per bucket.

### Syntax

```

float max_load_factor();
void max_load_factor(float new_factor);

```

### Parameters

*new\_factor*

New maximum load factor to store.

### Remarks

The first member function returns the current stored maximum load factor. You use it to determine the maximum average bucket size.

The second member function replaces the store maximum load factor with *new\_factor*. No automatic rehashing occurs until a subsequent insert.

### Example

```

// cliext_hash_multimap_max_load_factor.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1 = gcnew Myhash_multimap;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_multimap::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
hash_multimap<Key, Mapped>% operator=(hash_multimap<Key, Mapped>% right);
```

## Parameters

*right*

Container to copy.

## Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

## Example

```
// cliest_hash_multimap_operator_as.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliest::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Myhash_multimap c2;
    c2 = c1;
    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

## hash\_multimap::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

## Syntax

```
reverse_iterator rbegin();
```

## Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the `beginning` of the reverse sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

## Example

```

// cliext_hash_multimap_rbegin.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_multimap::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = [{0} {1}]\n",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("++rbegin() = [{0} {1}]\n",
        rit->first, rit->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*rbegin() = [c 3]
*++rbegin() = [b 2]

```

## hash\_multimap::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_hash_multimap_reference.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_multimap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
    {
        // get a reference to an element
        Myhash_multimap::reference ref = *it;
        System::Console::Write("[{0} {1}] ", ref->first, ref->second);
    }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## hash\_multimap::rehash (STL/CLR)

Rebuilds the hash table.

### Syntax

```
void rehash();
```

### Remarks

The member function rebuilds the hash table, ensuring that [hash\\_multimap::load\\_factor \(STL/CLR\)](#) ()  $\leq$  [hash\\_multimap::max\\_load\\_factor \(STL/CLR\)](#). Otherwise, the hash table increases in size only as needed after an insertion. (It never automatically decreases in size.) You use it to adjust the size of the hash table.

### Example

```

// cliext_hash_multimap_rehash.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1 = gcnew Myhash_multimap;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_multimap::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

## Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

## Example

```
// cliext_hash_multimap_rend.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_multimap::reverse_iterator rit = c1.rend();
    --rit;
    --rit;
    System::Console::WriteLine(" --- --rend() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine(" --- rend() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*-- --rend() = [b 2]
*--rend() = [a 1]
```

## hash\_multimap::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

## Syntax

```
typedef T3 reverse_iterator;
```

## Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

## Example

```
// cliext_hash_multimap_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Myhash_multimap::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("[{0} {1}] ", rit->first, rit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

## hash\_multimap::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [hash\\_multimap::empty \(STL/CLR\) \(\)](#).

### Example

```

// cliext_hash_multimap_size.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(Myhash_multimap::make_value(L'd', 4));
    c1.insert(Myhash_multimap::make_value(L'e', 5));
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0 after clearing
size() = 2 after adding 2

```

## hash\_multimap::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_hash_multimap_size_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_multimap::size_type diff = 0;
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3

```

## hash\_multimap::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```

void swap(hash_multimap<Key, Mapped>% right);

```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_hash_multimap_swap.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Myhash_multimap c2;
    c2.insert(Myhash_multimap::make_value(L'd', 4));
    c2.insert(Myhash_multimap::make_value(L'e', 5));
    c2.insert(Myhash_multimap::make_value(L'f', 6));
    for each (Myhash_multimap::value_type elem in c2)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    for each (Myhash_multimap::value_type elem in c2)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[d 4] [e 5] [f 6]
[d 4] [e 5] [f 6]
[a 1] [b 2] [c 3]

```

## hash\_multimap::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<value_type>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_hash_multimap_to_array.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // copy the container and modify it
    cli::array<Myhash_multimap::value_type>^ a1 = c1.to_array();

    c1.insert(Myhash_multimap::make_value(L'd', 4));
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (Myhash_multimap::value_type elem in a1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3]

```

## hash\_multimap::upper\_bound (STL/CLR)

Finds end of range that matches a specified key.

### Syntax

```

iterator upper_bound(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the last element *x* in the controlled sequence that hashes to the same bucket as *key* and has equivalent ordering to *key*. If no such element exists, or if *x* is the last element in the controlled sequence, it returns [hash\\_multimap::end \(STL/CLR\)](#)(); otherwise it returns an iterator that designates the first element beyond *x*. You use it to locate the end of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_hash_multimap_upper_bound.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    Myhash_multimap::iterator it = c1.upper_bound(L'a');
    System::Console::WriteLine("*upper_bound(L'a') = [{0} {1}]\n",
        it->first, it->second);
    it = c1.upper_bound(L'b');
    System::Console::WriteLine("*upper_bound(L'b') = [{0} {1}]\n",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
upper_bound(L'x') == end() = True
*upper_bound(L'a') = [b 2]
*upper_bound(L'b') = [c 3]

```

## hash\_multimap::value\_comp (STL/CLR)

Copies the ordering delegate for two element values.

### Syntax

```
value_compare^ value_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two element values.

### Example

```

// cliext_hash_multimap_value_comp.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Myhash_multimap::make_value(L'a', 1),
            Myhash_multimap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Myhash_multimap::make_value(L'a', 1),
            Myhash_multimap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Myhash_multimap::make_value(L'b', 2),
            Myhash_multimap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}

```

```

compare([L'a', 1], [L'a', 1]) = True
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False

```

## hash\_multimap::value\_compare (STL/CLR)

The ordering delegate for two element values.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
value_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its value arguments.

### Example

```

// cliext_hash_multimap_value_compare.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Myhash_multimap::make_value(L'a', 1),
            Myhash_multimap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Myhash_multimap::make_value(L'a', 1),
            Myhash_multimap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Myhash_multimap::make_value(L'b', 2),
            Myhash_multimap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}

```

```

compare([L'a', 1], [L'a', 1]) = True
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False

```

## hash\_multimap::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef generic_value value_type;
```

### Remarks

The type is a synonym for `generic_value`.

### Example

```
// cliext_hash_multimap_value_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using value_type
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Myhash_multimap::value_type val = *it;
        System::Console::Write("[{0} {1}] ", val->first, val->second);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

# hash\_multiset (STL/CLR)

9/20/2022 • 39 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `hash_multiset` to manage a sequence of elements as a hash table, each table entry storing a bidirectional linked list of nodes, and each node storing one element. The value of each element is used as a key, for ordering the sequence.

In the description below, `GValue` is the same as `GKey`, which in turn is the same as `Key` unless the latter is a ref type, in which case it is `Key^`.

## Syntax

```
template<typename Key>
ref class hash_multiset
    : public
        System::ICloneable,
        System::Collections::IEnumerable,
        System::Collections::ICollection,
        System::Collections::Generic::IEnumerable<GValue>,
        System::Collections::Generic::ICollection<GValue>,
        System::Collections::Generic::IList<GValue>,
        Microsoft::VisualC::StlClr::IHash<Gkey, GValue>
{ .....
```

### Parameters

#### Key

The type of the key component of an element in the controlled sequence.

## Requirements

**Header:** <cliext/hash\_set>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<code>hash_multiset::const_iterator (STL/CLR)</code>	The type of a constant iterator for the controlled sequence.
<code>hash_multiset::const_reference (STL/CLR)</code>	The type of a constant reference to an element.
<code>hash_multiset::const_reverse_iterator (STL/CLR)</code>	The type of a constant reverse iterator for the controlled sequence.
<code>hash_multiset::difference_type (STL/CLR)</code>	The type of a (possibly signed) distance between two elements.
<code>hash_multiset::generic_container (STL/CLR)</code>	The type of the generic interface for the container.

TYPE DEFINITION	DESCRIPTION
<code>hash_multiset::generic_iterator (STL/CLR)</code>	The type of an iterator for the generic interface for the container.
<code>hash_multiset::generic_reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the generic interface for the container.
<code>hash_multiset::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container.
<code>hash_multiset::hasher (STL/CLR)</code>	The hashing delegate for a key.
<code>hash_multiset::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>hash_multiset::key_compare (STL/CLR)</code>	The ordering delegate for two keys.
<code>hash_multiset::key_type (STL/CLR)</code>	The type of an ordering key.
<code>hash_multiset::reference (STL/CLR)</code>	The type of a reference to an element.
<code>hash_multiset::reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the controlled sequence.
<code>hash_multiset::size_type (STL/CLR)</code>	The type of a (non-negative) distance between two elements.
<code>hash_multiset::value_compare (STL/CLR)</code>	The ordering delegate for two element values.
<code>hash_multiset::value_type (STL/CLR)</code>	The type of an element.

MEMBER FUNCTION	DESCRIPTION
<code>hash_multiset::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>hash_multiset::bucket_count (STL/CLR)</code>	Counts the number of buckets.
<code>hash_multiset::clear (STL/CLR)</code>	Removes all elements.
<code>hash_multiset::count (STL/CLR)</code>	Counts elements matching a specified key.
<code>hash_multiset::empty (STL/CLR)</code>	Tests whether no elements are present.
<code>hash_multiset::end (STL/CLR)</code>	Designates the end of the controlled sequence.
<code>hash_multiset::equal_range (STL/CLR)</code>	Finds range that matches a specified key.
<code>hash_multiset::erase (STL/CLR)</code>	Removes elements at specified positions.
<code>hash_multiset::find (STL/CLR)</code>	Finds an element that matches a specified key.
<code>hash_multiset::hash_delegate (STL/CLR)</code>	Copies the hashing delegate for a key.
<code>hash_multiset::hash_multiset (STL/CLR)</code>	Constructs a container object.

MEMBER FUNCTION	DESCRIPTION
<a href="#">hash_multiset::insert (STL/CLR)</a>	Adds elements.
<a href="#">hash_multiset::key_comp (STL/CLR)</a>	Copies the ordering delegate for two keys.
<a href="#">hash_multiset::load_factor (STL/CLR)</a>	Counts the average elements per bucket.
<a href="#">hash_multiset::lower_bound (STL/CLR)</a>	Finds beginning of range that matches a specified key.
<a href="#">hash_multiset::make_value (STL/CLR)</a>	Constructs a value object.
<a href="#">hash_multiset::max_load_factor (STL/CLR)</a>	Gets or sets the maximum elements per bucket.
<a href="#">hash_multiset::rbegin (STL/CLR)</a>	Designates the beginning of the reversed controlled sequence.
<a href="#">hash_multiset::rehash (STL/CLR)</a>	Rebuilds the hash table.
<a href="#">hash_multiset::rend (STL/CLR)</a>	Designates the end of the reversed controlled sequence.
<a href="#">hash_multiset::size (STL/CLR)</a>	Counts the number of elements.
<a href="#">hash_multiset::swap (STL/CLR)</a>	Swaps the contents of two containers.
<a href="#">hash_multiset::to_array (STL/CLR)</a>	Copies the controlled sequence to a new array.
<a href="#">hash_multiset::upper_bound (STL/CLR)</a>	Finds end of range that matches a specified key.
<a href="#">hash_multiset::value_comp (STL/CLR)</a>	Copies the ordering delegate for two element values.
OPERATOR	DESCRIPTION
<a href="#">hash_multiset::operator= (STL/CLR)</a>	Replaces the controlled sequence.

## Interfaces

INTERFACE	DESCRIPTION
<a href="#">ICloneable</a>	Duplicate an object.
<a href="#">IEnumerable</a>	Sequence through elements.
<a href="#">ICollection</a>	Maintain group of elements.
<a href="#">IEnumerable&lt;T&gt;</a>	Sequence through typed elements.
<a href="#">ICollection&lt;T&gt;</a>	Maintain group of typed elements.
<a href="#">IHash&lt;Key, Value&gt;</a>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes in a bidirectional linked list. To speed access, the object also maintains a varying-length array of pointers into the list (the hash table), effectively managing the whole list as a sequence of sublists, or buckets. It inserts elements into a bucket that it keeps ordered by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements.

The object orders each bucket it controls by calling a stored delegate object of type [hash\\_set::key\\_compare \(STL/CLR\)](#). You can specify the stored delegate object when you construct the `hash_set`; if you specify no delegate object, the default is the comparison `operator<=(key_type, key_type)`.

You access the stored delegate object by calling the member function [hash\\_set::key\\_comp \(STL/CLR\) \(\)](#). Such a delegate object must define equivalent ordering between keys of type [hash\\_set::key\\_type \(STL/CLR\)](#). That means, for any two keys `x` and `y`:

`key_comp()(x, y)` returns the same Boolean result on every call.

If `key_comp()(x, y) && key_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

Any ordering rule that behaves like `operator<=(key_type, key_type)`, `operator>=(key_type, key_type)` or `operator==(key_type, key_type)` defines equivalent ordering.

Note that the container ensures only that elements whose keys have equivalent ordering (and which hash to the same integer value) are adjacent within a bucket. Unlike template class [hash\\_set \(STL/CLR\)](#), an object of template class [hash\\_multiset](#) does not require that keys for all elements are unique. (Two or more keys can have equivalent ordering.)

The object determines which bucket should contain a given ordering key by calling a stored delegate object of type [hash\\_set::hasher \(STL/CLR\)](#). You access this stored object by calling the member function [hash\\_set::hash\\_delegate \(STL/CLR\) \(\)](#) to obtain an integer value that depends on the key value. You can specify the stored delegate object when you construct the `hash_set`; if you specify no delegate object, the default is the function `System::Object::hash_value(key_type)`. That means, for any keys `x` and `y`:

`hash_delegate()(x)` returns the same integer result on every call.

If `x` and `y` have equivalent ordering, then `hash_delegate()(x)` should return the same integer result as `hash_delegate()(y)`.

Each element serves as both a key and a value. The sequence is represented in a way that permits lookup, insertion, and removal of an arbitrary element with a number of operations that is independent of the number of elements in the sequence (constant time) -- at least in the best of cases. Moreover, inserting an element invalidates no iterators, and removing an element invalidates only those iterators which point at the removed element.

If hashed values are not uniformly distributed, however, a hash table can degenerate. In the extreme -- for a hash function that always returns the same value -- lookup, insertion, and removal are proportional to the number of elements in the sequence (linear time). The container endeavors to choose a reasonable hash function, mean bucket size, and hash-table size (total number of buckets), but you can override any or all of these choices. See, for example, the functions [hash\\_set::max\\_load\\_factor \(STL/CLR\)](#) and [hash\\_set::rehash \(STL/CLR\)](#).

A `hash_multiset` supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by [hash\\_multiset::end \(STL/CLR\) \(\)](#). You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a `hash_multiset` iterator to reach the head node, and it will then compare equal to `end()`. But you cannot dereference the iterator returned by `end()`.

Note that you cannot refer to a hash\_multiset element directly given its numerical position -- that requires a random-access iterator.

A hash\_multiset iterator stores a handle to its associated hash\_multiset node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A hash\_multiset iterator remains valid so long as its associated hash\_multiset node is associated with some hash\_multiset. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### hash\_multiset::begin (STL/CLR)

Designates the beginning of the controlled sequence.

#### Syntax

```
iterator begin();
```

#### Remarks

The member function returns a bidirectional iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

#### Example

```
// cliext_hash_multiset_begin.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_multiset::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("**begin() = {0}", *++it);
    return (0);
}
```

```
a b c
*begin() = a
*++begin() = b
```

## hash\_multiset::bucket\_count (STL/CLR)

Counts the number of buckets.

### Syntax

```
int bucket_count();
```

### Remarks

The member functions returns the current number of buckets. You use it to determine the size of the hash table.

### Example

```
// cliext_hash_multiset_bucket_count.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}
```

```
a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25
```

## hash\_multiset::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

### Remarks

The member function effectively calls [hash\\_multiset::erase \(STL/CLR\) \(](#) [hash\\_multiset::begin \(STL/CLR\) \(\)](#), [hash\\_multiset::end \(STL/CLR\) \(\)](#) ). You use it to ensure that the controlled sequence is empty.

### Example

```
// cliext_hash_multiset_clear.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
size() = 0
a b
size() = 0
```

## hash\_multiset::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

### Syntax

```
typedef T2 const_iterator;
```

### Remarks

The type describes an object of unspecified type `T2` that can serve as a constant bidirectional iterator for the controlled sequence.

### Example

```
// cliext_hash_multiset_const_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myhash_multiset::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## hash\_multiset::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_hash_multiset_const_reference.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myhash_multiset::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Myhash_multiset::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## hash\_multiset::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```

// cliext_hash_multiset_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myhash_multiset::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}

```

```
c b a
```

## hash\_multiset::count (STL/CLR)

Finds the number of elements matching a specified key.

### Syntax

```
size_type count(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns the number of elements in the controlled sequence that have equivalent ordering with *key*. You use it to determine the number of elements currently in the controlled sequence that match a specified key.

### Example

```
// cliext_hash_multiset_count.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
a b c
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

## hash\_multiset::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

## Remarks

The type describes a possibly negative element count.

## Example

```
// cliext_hash_multiset_difference_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_multiset::difference_type diff = 0;
    for (Myhash_multiset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Myhash_multiset::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
begin()-end() = -3
```

## hash\_multiset::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

## Remarks

The member function returns true for an empty controlled sequence. It is equivalent to `hash_multiset::size` (STL/CLR) `() == 0`. You use it to test whether the hash\_multiset is empty.

## Example

```

// cliext_hash_multiset_empty.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

## hash\_multiset::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a bidirectional iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_hash_multiset_end.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    Myhash_multiset::iterator it = c1.end();
    --it;
    System::Console::WriteLine("--- --end() = {0}", *--it);
    System::Console::WriteLine("---end() = {0}", *++it);
    return (0);
}

```

```

a b c
--- --end() = b
---end() = c

```

## hash\_multiset::equal\_range (STL/CLR)

Finds range that matches a specified key.

### Syntax

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns a pair of iterators `cliext::pair<iterator, iterator>( hash_multiset::lower_bound (STL/CLR) (key), hash_multiset::upper_bound (STL/CLR) (key))`. You use it to determine the range of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_hash_multiset_equal_range.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
typedef Myhash_multiset::pair_iter_iter Pairii;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("{0} ", *pair1.first);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
equal_range(L'x') empty = True
b

```

## hash\_multiset::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

### Parameters

*first*

Beginning of range to erase.

*key*

Key value to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*, and returns an

iterator that designates the first element remaining beyond the element removed, or `hash_multiset::end` (**STL/CLR**) if no such element exists. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [`first`, `last`), and returns an iterator that designates the first element remaining beyond any elements removed, or `end()` if no such element exists.. You use it to remove zero or more contiguous elements.

The third member function removes any element of the controlled sequence whose key has equivalent ordering to `key`, and returns a count of the number of elements removed. You use it to remove and count all elements that match a specified key.

Each element erasure takes time proportional to the logarithm of the number of elements in the controlled sequence.

## Example

```
// cliext_hash_multiset_erase.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.insert(L'd');
    c1.insert(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    Myhash_multiset::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

## hash\_multiset::find (STL/CLR)

Finds an element that matches a specified key.

## Syntax

```
iterator find(key_type key);
```

## Parameters

### key

Key value to search for.

## Remarks

If at least one element in the controlled sequence has equivalent ordering with *key*, the member function returns an iterator designating one of those elements; otherwise it returns [hash\\_multiset::end \(STL/CLR\)](#) (). You use it to locate an element currently in the controlled sequence that matches a specified key.

## Example

```
// cliext_hash_multiset_find.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());
    System::Console::WriteLine("find {0} = {1}",
        L'b', *c1.find(L'b'));
    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
a b c
find A = False
find b = b
find C = False
```

## hash\_multiset::generic\_container (STL/CLR)

The type of the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::
    IHash<GKey, GValue>
generic_container;
```

## Remarks

The type describes the generic interface for this template container class.

## Example

```
// cliext_hash_multiset_generic_container.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(L'e');
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
a b c d
a b c d e
```

## hash\_multiset::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

## Example

```

// cliext_hash_multiset_generic_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_multiset::generic_iterator gcit = gc1->begin();
    Myhash_multiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

## hash\_multiset::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value>
generic_reverse_iterator;

```

### Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_hash_multiset_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_multiset::generic_reverse_iterator gcit = gc1->rbegin();
    Myhash_multiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
c

```

## hash\_multiset::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```

typedef GValue generic_value;

```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

### Example

```

// cliext_hash_multiset_generic_value.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_multiset::generic_iterator gcit = gc1->begin();
    Myhash_multiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

## hash\_multiset::hash\_delegate (STL/CLR)

Finds an element that matches a specified key.

### Syntax

```

hasher^ hash_delegate();

```

### Remarks

The member function returns the delegate used to convert a key value to an integer. You use it to hash a key.

### Example

```

// cliext_hash_multiset_hash_delegate.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}

```

```

hash(L'a') = 1616896120
hash(L'b') = 570892832

```

## hash\_multiset::hash\_multiset (STL/CLR)

Constructs a container object.

### Syntax

```

hash_multiset();
explicit hash_multiset(key_compare^ pred);
hash_multiset(key_compare^ pred, hasher^ hashfn);
hash_multiset(hash_multiset<Key>% right);
hash_multiset(hash_multiset<Key>^ right);
template<typename InIter>
    hash_multiset(InIter first, InIter last);
template<typename InIter>
    hash_multiset(InIter first, InIter last,
        key_compare^ pred);
template<typename InIter>
    hash_multiset(InIter first, InIter last,
        key_compare^ pred, hasher^ hashfn);
hash_multiset(System::Collections::Generic::IEnumerable<GValue>^ right);
hash_multiset(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);
hash_multiset(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred, hasher^ hashfn);

```

### Parameters

*first*

Beginning of range to insert.

*hashfn*

Hash function for mapping keys to buckets.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

### Remarks

The constructor:

```
hash_multiset();
```

initializes the controlled sequence with no elements, with the default ordering predicate `key_compare()`, and with the default hash function. You use it to specify an empty initial controlled sequence, with the default ordering predicate and hash function.

The constructor:

```
explicit hash_multiset(key_compare^ pred);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*, and with the default hash function. You use it to specify an empty initial controlled sequence, with the specified ordering predicate and the default hash function.

The constructor:

```
hash_multiset(key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*, and with the hash function *hashfn*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate and hash function.

The constructor:

```
hash_multiset(hash_multiset<Key>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`], with the default ordering predicate, and with the default hash function. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `hash_multiset` object *right*, with the default ordering predicate and hash function.

The constructor:

```
hash_multiset(hash_multiset<Key>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`], with the default ordering predicate, and with the default hash function. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `hash_multiset` object *right*, with the default ordering predicate and hash function.

The constructor:

```
template<typename InIter> hash_multiset(InIter first, InIter last);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the default ordering predicate, and with the default hash function. You use it to make the controlled sequence a copy of another sequence, with the default ordering predicate and hash function.

The constructor:

```
template<typename InIter> hash_multiset(InIter first, InIter last, key_compare^ pred);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*, and with the default hash function. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate and the default hash function.

The constructor:

```
template<typename InIter> hash_multiset(InIter first, InIter last, key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*, and with

the hash function *hashfn*. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate and hash function.

The constructor:

```
hash_multiset(System::Collections::Generic::IEnumerable<Key>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the default ordering predicate, and with the default hash function. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the default ordering predicate and hash function.

The constructor:

```
hash_multiset(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*, and with the default hash function. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate and default hash function.

The constructor:

```
hash_multiset(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*, and with the hash function *hashfn*. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate and hash function.

## Example

```
// cliext_hash_multiset_construct.cpp
// compile with: /clr
#include <cliext/hash_set>

int myfun(wchar_t key)
{ // hash a key
    return (key ^ 0xdeadbeef);
}

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    //
    // construct an empty container
    Myhash_multiset c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an ordering rule
    Myhash_multiset c2 = cliext::greater_equal<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    c2.insert(c1.begin(), c1.end());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an ordering rule and hash function
    Myhash_multiset c2h(cliext::greater_equal<wchar_t>(),
        gcnew Myhash_multiset::hasher(&myfun));
    System::Console::WriteLine("size() = {0}", c2h.size());
```

```

c2h.insert(c1.begin(), c1.end());
for each (wchar_t elem in c2h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an iterator range
Myhash_multiset c3(c1.begin(), c1.end());
for each (wchar_t elem in c3)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Myhash_multiset c4(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c4)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule and hash function
Myhash_multiset c4h(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_multiset::hasher(&myfun));
for each (wchar_t elem in c4h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an enumeration
Myhash_multiset c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
for each (wchar_t elem in c5)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myhash_multiset c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c6)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule and hash function
Myhash_multiset c6h( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_multiset::hasher(&myfun));
for each (wchar_t elem in c6h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct from a generic container
Myhash_multiset c7(c4);
for each (wchar_t elem in c7)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct by copying another container
Myhash_multiset c8(%c3);
for each (wchar_t elem in c8)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
a b c
size() = 0
a b c
size() = 0
c b a

a b c
a b c
c b a

a b c
a b c
c b a

a b c
a b c
```

## hash\_multiset::hasher (STL/CLR)

The hashing delegate for a key.

### Syntax

```
Microsoft::VisualC::StlClr::UnaryDelegate<GKey, int>
    hasher;
```

### Remarks

The type describes a delegate that converts a key value to an integer.

### Example

```
// cliext_hash_multiset_hasher.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

## hash\_multiset::insert (STL/CLR)

Adds elements.

### Syntax

```
iterator insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumarable<value_type>^ right);
```

## Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Key value to insert.

*where*

Where in container to insert (hint only).

## Remarks

Each of the member functions inserts a sequence specified by the remaining operands.

The first member function inserts an element with value *val*, and returns an iterator that designates the newly inserted element. You use it to insert a single element.

The second member function inserts an element with value *val*, using *where* as a hint (to improve performance), and returns an iterator that designates the newly inserted element. You use it to insert a single element which might be adjacent to an element you know.

The third member function inserts the sequence [*first*, *last*). You use it to insert zero or more elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

Each element insertion takes time proportional to the logarithm of the number of elements in the controlled sequence. Insertion can occur in amortized constant time, however, given a hint that designates an element adjacent to the insertion point.

## Example

```

// cliext_hash_multiset_insert.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    System::Console::WriteLine("insert(L'x') = {0}",
        *c1.insert(L'x'));

    System::Console::WriteLine("insert(L'b') = {0}",
        *c1.insert(L'b'));

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value with hint
    System::Console::WriteLine("insert(begin(), L'y') = {0}",
        *c1.insert(c1.begin(), L'y'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    Myhash_multiset c2;
    Myhash_multiset::iterator it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    Myhash_multiset c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)c1);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(L'x') = x
insert(L'b') = b
a b b c x
insert(begin(), L'y') = y
a b b c x y
a b b c x
a b b c x y

```

## hash\_multiset::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

## Syntax

```
typedef T1 iterator;
```

## Remarks

The type describes an object of unspecified type `T1` that can serve as a bidirectional iterator for the controlled sequence.

## Example

```
// cliext_hash_multiset_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myhash_multiset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## hash\_multiset::key\_comp (STL/CLR)

Copies the ordering delegate for two keys.

## Syntax

```
key_compare^key_comp();
```

## Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two keys.

## Example

```

// cliext_hash_multiset_key_comp.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_multiset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## hash\_multiset::key\_compare (STL/CLR)

The ordering delegate for two keys.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its key arguments.

### Example

```

// cliext_hash_multiset_key_compare.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_multiset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## hash\_multiset::key\_type (STL/CLR)

The type of an ordering key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter *Key*.

### Example

```

// cliext_hash_multiset_key_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using key_type
    for (Myhash_multiset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myhash_multiset::key_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## hash\_multiset::load\_factor (STL/CLR)

Counts the average elements per bucket.

### Syntax

```
float load_factor();
```

### Remarks

The member function returns `(float) hash_multiset::size (STL/CLR) () / hash_multiset::bucket_count (STL/CLR) ()`. You use it to determine the average bucket size.

### Example

```

// cliext_hash_multiset_load_factor.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_multiset::lower\_bound (STL/CLR)

Finds beginning of range that matches a specified key.

### Syntax

```
iterator lower_bound(key_type key);
```

## Parameters

*key*

Key value to search for.

## Remarks

The member function determines the first element *x* in the controlled sequence that hashes to the same bucket as *key* and has equivalent ordering to *key*. If no such element exists, it returns [hash\\_multiset::end \(STL/CLR\)](#) (); otherwise it returns an iterator that designates *x*. You use it to locate the beginning of a sequence of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_hash_multiset_lower_bound.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    System::Console::WriteLine("*lower_bound(L'a') = {0}",
        *c1.lower_bound(L'a'));
    System::Console::WriteLine("*lower_bound(L'b') = {0}",
        *c1.lower_bound(L'b'));
    return (0);
}
```

```
a b c
lower_bound(L'x') == end() = True
*lower_bound(L'a') = a
*lower_bound(L'b') = b
```

## hash\_multiset::make\_value (STL/CLR)

Constructs a value object.

## Syntax

```
static value_type make_value(key_type key);
```

## Parameters

*key*

Key value to use.

## Remarks

The member function returns a *value\_type* object whose key is *key*. You use it to compose an object suitable for use with several other member functions.

## Example

```
// cliext_hash_multiset_make_value.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(Myhash_multiset::make_value(L'a'));
    c1.insert(Myhash_multiset::make_value(L'b'));
    c1.insert(Myhash_multiset::make_value(L'c'));

    // display contents " a b c"
    for each (Myhash_multiset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## hash\_multiset::max\_load\_factor (STL/CLR)

Gets or sets the maximum elements per bucket.

### Syntax

```
float max_load_factor();
void max_load_factor(float new_factor);
```

### Parameters

*new\_factor*

New maximum load factor to store.

### Remarks

The first member function returns the current stored maximum load factor. You use it to determine the maximum average bucket size.

The second member function replaces the store maximum load factor with *new\_factor*. No automatic rehashing occurs until a subsequent insert.

## Example

```

// cliext_hash_multiset_max_load_factor.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_multiset::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
hash_multiset<Key>% operator=(hash_multiset<Key>% right);
```

## Parameters

*right*

Container to copy.

## Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

## Example

```
// cliext_hash_multiset_operator_as.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (Myhash_multiset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myhash_multiset c2;
    c2 = c1;
    // display contents " a b c"
    for each (Myhash_multiset::value_type elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

## hash\_multiset::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

## Syntax

```
reverse_iterator rbegin();
```

## Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the `beginning` of the reverse sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

## Example

```

// cliext_hash_multiset_rbegin.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_multiset::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("rbegin() = {0}", *rit);
    System::Console::WriteLine("++rbegin() = {0}", ++rit);
    return (0);
}

```

```

a b c
*rbegin() = c
*++rbegin() = b

```

## hash\_multiset::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```

typedef value_type% reference;

```

### Remarks

The type describes a reference to an element.

### Example

```
// cliext_hash_multiset_reference.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myhash_multiset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        Myhash_multiset::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## hash\_multiset::rehash (STL/CLR)

Rebuilds the hash table.

### Syntax

```
void rehash();
```

### Remarks

The member function rebuilds the hash table, ensuring that [hash\\_multiset::load\\_factor \(STL/CLR\) \(\)](#)  $\leq$  [hash\\_multiset::max\\_load\\_factor \(STL/CLR\)](#). Otherwise, the hash table increases in size only as needed after an insertion. (It never automatically decreases in size.) You use it to adjust the size of the hash table.

### Example

```

// cliext_hash_multiset_rehash.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_multiset::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

## Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

## Example

```
// cliext_hash_multiset_rend.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_multiset::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("--- --rend() = {0}", *--rit);
    System::Console::WriteLine("---rend() = {0}", *++rit);
    return (0);
}
```

```
a b c
--- --rend() = b
---rend() = a
```

## hash\_multiset::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

## Syntax

```
typedef T3 reverse_iterator;
```

## Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

## Example

```
// cliext_hash_multiset_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myhash_multiset::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## hash\_multiset::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [hash\\_multiset::empty \(STL/CLR\) \(\)](#).

### Example

```

// cliext_hash_multiset_size.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

## hash\_multiset::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_hash_multiset_size_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_multiset::size_type diff = 0;
    for (Myhash_multiset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3

```

## hash\_multiset::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```

void swap(hash_multiset<Key>% right);

```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_hash_multiset_swap.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Myhash_multiset c2;
    c2.insert(L'd');
    c2.insert(L'e');
    c2.insert(L'f');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
d e f
d e f
a b c

```

## hash\_multiset::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<value_type>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_hash_multiset_to_array.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c d
a b c

```

## hash\_multiset::upper\_bound (STL/CLR)

Finds end of range that matches a specified key.

### Syntax

```

iterator upper_bound(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the last element *x* in the controlled sequence that hashes to the same bucket as *key* and has equivalent ordering to *key*. If no such element exists, or if *x* is the last element in the controlled sequence, it returns [hash\\_multiset::end \(STL/CLR\)](#); otherwise it returns an iterator that designates the first element beyond *x*. You use it to locate the end of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_hash_multiset_upper_bound.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    System::Console::WriteLine("*upper_bound(L'a') = {0}",
        *c1.upper_bound(L'a'));
    System::Console::WriteLine("*upper_bound(L'b') = {0}",
        *c1.upper_bound(L'b'));
    return (0);
}

```

```

a b c
upper_bound(L'x') == end() = True
*upper_bound(L'a') = b
*upper_bound(L'b') = c

```

## hash\_multiset::value\_comp (STL/CLR)

Copies the ordering delegate for two element values.

### Syntax

```

value_compare^ value_comp();

```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two element values.

### Example

```

// cliext_hash_multiset_value_comp.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

```

## hash\_multiset::value\_compare (STL/CLR)

The ordering delegate for two element values.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its value arguments.

### Example

```

// cliext_hash_multiset_value_compare.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```
compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

## hash\_multiset::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef generic_value value_type;
```

### Remarks

The type is a synonym for `generic_value`.

### Example

```
// cliext_hash_multiset_value_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using value_type
    for (Myhash_multiset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Myhash_multiset::value_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

# hash\_set (STL/CLR)

9/20/2022 • 38 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `hash_set` to manage a sequence of elements as a hash table, each table entry storing a bidirectional linked list of nodes, and each node storing one element. The value of each element is used as a key, for ordering the sequence.

In the description below, `GValue` is the same as `GKey`, which in turn is the same as `Key` unless the latter is a ref type, in which case it is `Key^`.

## Syntax

```
template<typename Key>
ref class hash_set
    : public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    Microsoft::VisualC::StlClr::IHash<Gkey, GValue>
{ .....
```

### Parameters

#### Key

The type of the key component of an element in the controlled sequence.

## Requirements

**Header:** <cliext/hash\_set>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">hash_set::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.
<a href="#">hash_set::const_reference (STL/CLR)</a>	The type of a constant reference to an element.
<a href="#">hash_set::const_reverse_iterator (STL/CLR)</a>	The type of a constant reverse iterator for the controlled sequence.
<a href="#">hash_set::difference_type (STL/CLR)</a>	The type of a (possibly signed) distance between two elements.
<a href="#">hash_set::generic_container (STL/CLR)</a>	The type of the generic interface for the container.

TYPE DEFINITION	DESCRIPTION
<a href="#">hash_set::generic_iterator (STL/CLR)</a>	The type of an iterator for the generic interface for the container.
<a href="#">hash_set::generic_reverse_iterator (STL/CLR)</a>	The type of a reverse iterator for the generic interface for the container.
<a href="#">hash_set::generic_value (STL/CLR)</a>	The type of an element for the generic interface for the container.
<a href="#">hash_set::hasher (STL/CLR)</a>	The hashing delegate for a key.
<a href="#">hash_set::iterator (STL/CLR)</a>	The type of an iterator for the controlled sequence.
<a href="#">hash_set::key_compare (STL/CLR)</a>	The ordering delegate for two keys.
<a href="#">hash_set::key_type (STL/CLR)</a>	The type of an ordering key.
<a href="#">hash_set::reference (STL/CLR)</a>	The type of a reference to an element.
<a href="#">hash_set::reverse_iterator (STL/CLR)</a>	The type of a reverse iterator for the controlled sequence.
<a href="#">hash_set::size_type (STL/CLR)</a>	The type of a (non-negative) distance between two elements.
<a href="#">hash_set::value_compare (STL/CLR)</a>	The ordering delegate for two element values.
<a href="#">hash_set::value_type (STL/CLR)</a>	The type of an element.

MEMBER FUNCTION	DESCRIPTION
<a href="#">hash_set::begin (STL/CLR)</a>	Designates the beginning of the controlled sequence.
<a href="#">hash_set::bucket_count (STL/CLR)</a>	Counts the number of buckets.
<a href="#">hash_set::clear (STL/CLR)</a>	Removes all elements.
<a href="#">hash_set::count (STL/CLR)</a>	Counts elements matching a specified key.
<a href="#">hash_set::empty (STL/CLR)</a>	Tests whether no elements are present.
<a href="#">hash_set::end (STL/CLR)</a>	Designates the end of the controlled sequence.
<a href="#">hash_set::equal_range (STL/CLR)</a>	Finds range that matches a specified key.
<a href="#">hash_set::erase (STL/CLR)</a>	Removes elements at specified positions.
<a href="#">hash_set::find (STL/CLR)</a>	Finds an element that matches a specified key.
<a href="#">hash_set::hash_delegate (STL/CLR)</a>	Copies the hashing delegate for a key.
<a href="#">hash_set::hash_set (STL/CLR)</a>	Constructs a container object.

MEMBER FUNCTION	DESCRIPTION
<code>hash_set::insert (STL/CLR)</code>	Adds elements.
<code>hash_set::key_comp (STL/CLR)</code>	Copies the ordering delegate for two keys.
<code>hash_set::load_factor (STL/CLR)</code>	Counts the average elements per bucket.
<code>hash_set::lower_bound (STL/CLR)</code>	Finds beginning of range that matches a specified key.
<code>hash_set::make_value (STL/CLR)</code>	Constructs a value object.
<code>hash_set::max_load_factor (STL/CLR)</code>	Gets or sets the maximum elements per bucket.
<code>hash_set::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.
<code>hash_set::rehash (STL/CLR)</code>	Rebuilds the hash table.
<code>hash_set::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>hash_set::size (STL/CLR)</code>	Counts the number of elements.
<code>hash_set::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>hash_set::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>hash_set::upper_bound (STL/CLR)</code>	Finds end of range that matches a specified key.
<code>hash_set::value_comp (STL/CLR)</code>	Copies the ordering delegate for two element values.
OPERATOR	DESCRIPTION
<code>hash_set::operator= (STL/CLR)</code>	Replaces the controlled sequence.

## Interfaces

INTERFACE	DESCRIPTION
<code>ICloneable</code>	Duplicate an object.
<code>IEnumerable</code>	Sequence through elements.
<code>ICollection</code>	Maintain group of elements.
<code>IEnumerable&lt;T&gt;</code>	Sequence through typed elements.
<code>ICollection&lt;T&gt;</code>	Maintain group of typed elements.
<code>IHash&lt;Key, Value&gt;</code>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes in a bidirectional linked list. To speed access, the object also maintains a varying-length array of pointers into the list (the hash table), effectively managing the whole list as a sequence of sublists, or buckets. It inserts elements into a bucket that it keeps ordered by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements.

The object orders each bucket it controls by calling a stored delegate object of type [hash\\_set::key\\_compare \(STL/CLR\)](#). You can specify the stored delegate object when you construct the `hash_set`; if you specify no delegate object, the default is the comparison `operator<=(key_type, key_type)`.

You access the stored delegate object by calling the member function [hash\\_set::key\\_comp \(STL/CLR\) \(\)](#). Such a delegate object must define equivalent ordering between keys of type [hash\\_set::key\\_type \(STL/CLR\)](#). That means, for any two keys `x` and `y`:

`key_comp()(x, y)` returns the same Boolean result on every call.

If `key_comp()(x, y) && key_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

Any ordering rule that behaves like `operator<=(key_type, key_type)`, `operator>=(key_type, key_type)` or `operator==(key_type, key_type)` defines equivalent ordering.

Note that the container ensures only that elements whose keys have equivalent ordering (and which hash to the same integer value) are adjacent within a bucket. Unlike template class [hash\\_multiset \(STL/CLR\)](#), an object of template class `hash_set` ensures that keys for all elements are unique. (No two keys have equivalent ordering.)

The object determines which bucket should contain a given ordering key by calling a stored delegate object of type [hash\\_set::hasher \(STL/CLR\)](#). You access this stored object by calling the member function [hash\\_set::hash\\_delegate \(STL/CLR\) \(\)](#) to obtain an integer value that depends on the key value. You can specify the stored delegate object when you construct the `hash_set`; if you specify no delegate object, the default is the function `System::Object::hash_value(key_type)`. That means, for any keys `x` and `y`:

`hash_delegate()(x)` returns the same integer result on every call.

If `x` and `y` have equivalent ordering, then `hash_delegate()(x)` should return the same integer result as `hash_delegate()(y)`.

Each element serves as both a key and a value. The sequence is represented in a way that permits lookup, insertion, and removal of an arbitrary element with a number of operations that is independent of the number of elements in the sequence (constant time) -- at least in the best of cases. Moreover, inserting an element invalidates no iterators, and removing an element invalidates only those iterators which point at the removed element.

If hashed values are not uniformly distributed, however, a hash table can degenerate. In the extreme -- for a hash function that always returns the same value -- lookup, insertion, and removal are proportional to the number of elements in the sequence (linear time). The container endeavors to choose a reasonable hash function, mean bucket size, and hash-table size (total number of buckets), but you can override any or all of these choices. See, for example, the functions [hash\\_set::max\\_load\\_factor \(STL/CLR\)](#) and [hash\\_set::rehash \(STL/CLR\)](#).

A `hash_set` supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by [hash\\_set::end \(STL/CLR\) \(\)](#). You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a `hash_set` iterator to reach the head node, and it will then compare equal to `end()`. But you cannot dereference the iterator returned by `end()`.

Note that you cannot refer to a `hash_set` element directly given its numerical position -- that requires a random-

access iterator.

A hash\_set iterator stores a handle to its associated hash\_set node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A hash\_set iterator remains valid so long as its associated hash\_set node is associated with some hash\_set. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### hash\_set::begin (STL/CLR)

Designates the beginning of the controlled sequence.

#### Syntax

```
iterator begin();
```

#### Remarks

The member function returns a bidirectional iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

#### Example

```
// cliext_hash_set_begin.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    {
        Myhash_Set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect first two items
        Myhash_Set::iterator it = c1.begin();
        System::Console::WriteLine("begin() = {0}", *it);
        System::Console::WriteLine("++begin() = {0}", *++it);
        return (0);
    }
}
```

### hash\_set::bucket\_count (STL/CLR)

Counts the number of buckets.

## Syntax

```
int bucket_count();
```

## Remarks

The member functions returns the current number of buckets. You use it to determine the size of the hash table.

## Example

```
// cliext_hash_set_bucket_count.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}
```

```
a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25
```

## hash\_set::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

### Remarks

The member function effectively calls [hash\\_set::erase \(STL/CLR\)](#) ( [hash\\_set::begin \(STL/CLR\)](#) (), [hash\\_set::end \(STL/CLR\)](#) () ). You use it to ensure that the controlled sequence is empty.

### Example

```
// cliext_hash_set_clear.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
size() = 0
a b
size() = 0
```

## hash\_set::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

### Syntax

```
typedef T2 const_iterator;
```

## Remarks

The type describes an object of unspecified type `T2` that can serve as a constant bidirectional iterator for the controlled sequence.

## Example

```
// cliext_hash_set_const_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myhash_Set::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## hash\_set::const\_reference (STL/CLR)

The type of a constant reference to an element.

## Syntax

```
typedef value_type% const_reference;
```

## Remarks

The type describes a constant reference to an element.

## Example

```

// cliext_hash_set_const_reference.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myhash_Set::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Myhash_Set::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## hash\_set::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```

// cliext_hash_set_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myhash_Set::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}

```

```
c b a
```

## hash\_set::count (STL/CLR)

Finds the number of elements matching a specified key.

### Syntax

```
size_type count(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns the number of elements in the controlled sequence that have equivalent ordering with *key*. You use it to determine the number of elements currently in the controlled sequence that match a specified key.

### Example

```
// cliext_hash_set_count.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
a b c
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

## hash\_set::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

## Remarks

The type describes a possibly negative element count.

## Example

```
// cliext_hash_set_difference_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_Set::difference_type diff = 0;
    for (Myhash_Set::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Myhash_Set::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
begin()-end() = -3
```

## hash\_set::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

## Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [hash\\_set::size \(STL/CLR\) \(\) == 0](#). You use it to test whether the hash\_set is empty.

## Example

```

// cliext_hash_set_empty.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

## hash\_set::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a bidirectional iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_hash_set_end.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    Myhash_Set::iterator it = c1.end();
    --it;
    System::Console::WriteLine("--- --end() = {0}", *--it);
    System::Console::WriteLine("---end() = {0}", *++it);
    return (0);
}

```

```

a b c
--- --end() = b
---end() = c

```

## hash\_set::equal\_range (STL/CLR)

Finds range that matches a specified key.

### Syntax

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

### Parameters

#### *key*

Key value to search for.

### Remarks

The member function returns a pair of iterators `cliext::pair<iterator, iterator>( hash_set::lower_bound (STL/CLR) (key), hash_set::upper_bound (STL/CLR) (key))`. You use it to determine the range of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_hash_set_equal_range.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
typedef Myhash_Set::pair_iter_iter Pairii;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("{0} ", *pair1.first);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
equal_range(L'x') empty = True
b

```

## hash\_set::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

### Parameters

*first*

Beginning of range to erase.

*key*

Key value to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*, and returns an

iterator that designates the first element remaining beyond the element removed, or [hash\\_set::end \(STL/CLR\)](#)

(*)* if no such element exists. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [`first`, `last`), and returns an iterator that designates the first element remaining beyond any elements removed, or `end()` if no such element exists.. You use it to remove zero or more contiguous elements.

The third member function removes any element of the controlled sequence whose key has equivalent ordering to `key`, and returns a count of the number of elements removed. You use it to remove and count all elements that match a specified key.

Each element erasure takes time proportional to the logarithm of the number of elements in the controlled sequence.

## Example

```
// cliext_hash_set_erase.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.insert(L'd');
    c1.insert(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    Myhash_set::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

## hash\_set::find (STL/CLR)

Finds an element that matches a specified key.

## Syntax

```
iterator find(key_type key);
```

## Parameters

### key

Key value to search for.

## Remarks

If at least one element in the controlled sequence has equivalent ordering with *key*, the member function returns an iterator designating one of those elements; otherwise it returns [hash\\_set::end \(STL/CLR\)](#) (). You use it to locate an element currently in the controlled sequence that matches a specified key.

## Example

```
// cliext_hash_set_find.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());
    System::Console::WriteLine("find {0} = {1}",
        L'b', *c1.find(L'b'));
    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
a b c
find A = False
find b = b
find C = False
```

## hash\_set::generic\_container (STL/CLR)

The type of the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::
    IHash<GKey, GValue>
generic_container;
```

## Remarks

The type describes the generic interface for this template container class.

## Example

```
// cliext_hash_set_generic_container.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_set::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(L'e');
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
a b c d
a b c d e
```

## hash\_set::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

## Example

```

// cliext_hash_set_generic_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_Set::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_Set::generic_iterator gcit = gc1->begin();
    Myhash_Set::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

## hash\_set::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value>
generic_reverse_iterator;

```

### Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_hash_set_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_Set::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_Set::generic_reverse_iterator gcit = gc1->rbegin();
    Myhash_Set::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
c

```

## hash\_set::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```

typedef GValue generic_value;

```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

### Example

```

// cliext_hash_set_generic_value.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_Set::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_Set::generic_iterator gcit = gc1->begin();
    Myhash_Set::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

## hash\_set::hash\_delegate (STL/CLR)

Finds an element that matches a specified key.

### Syntax

```

hasher^ hash_delegate();

```

### Remarks

The member function returns the delegate used to convert a key value to an integer. You use it to hash a key.

### Example

```

// cliext_hash_set_hash_delegate.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}

```

```

hash(L'a') = 1616896120
hash(L'b') = 570892832

```

## hash\_set::hash\_set (STL/CLR)

Constructs a container object.

### Syntax

```

hash_set();
explicit hash_set(key_compare^ pred);
hash_set(key_compare^ pred, hasher^ hashfn);
hash_set(hash_set<Key>% right);
hash_set(hash_set<Key>^ right);
template<typename InIter>
    hash_set(hash_set<InIter first, InIter last>);
template<typename InIter>
    hash_set(InIter first, InIter last,
             key_compare^ pred);
template<typename InIter>
    hash_set(InIter first, InIter last,
             key_compare^ pred, hasher^ hashfn);
hash_set(System::Collections::Generic::IEnumerable<GValue>^ right);
hash_set(System::Collections::Generic::IEnumerable<GValue>^ right,
         key_compare^ pred);
hash_set(System::Collections::Generic::IEnumerable<GValue>^ right,
         key_compare^ pred, hasher^ hashfn);

```

### Parameters

*first*

Beginning of range to insert.

*hashfn*

Hash function for mapping keys to buckets.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

### Remarks

The constructor:

```
hash_set();
```

initializes the controlled sequence with no elements, with the default ordering predicate `key_compare()`, and with the default hash function. You use it to specify an empty initial controlled sequence, with the default ordering predicate and hash function.

The constructor:

```
explicit hash_set(key_compare^ pred);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*, and with the default hash function. You use it to specify an empty initial controlled sequence, with the specified ordering predicate and the default hash function.

The constructor:

```
hash_set(key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*, and with the hash function *hashfn*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate and hash function.

The constructor:

```
hash_set(hash_set<Key>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`], with the default ordering predicate, and with the default hash function. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `hash_set` object *right*, with the default ordering predicate and hash function.

The constructor:

```
hash_set(hash_set<Key>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`], with the default ordering predicate, and with the default hash function. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `hash_set` object *right*, with the default ordering predicate and hash function.

The constructor:

```
template<typename InIter> hash_set(InIter first, InIter last);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the default ordering predicate, and with the default hash function. You use it to make the controlled sequence a copy of another sequence, with the default ordering predicate and hash function.

The constructor:

```
template<typename InIter> hash_set(InIter first, InIter last, key_compare^ pred);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*, and with the default hash function. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate and the default hash function.

The constructor:

```
template<typename InIter> hash_set(InIter first, InIter last, key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*, and with

the hash function *hashfn*. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate and hash function.

The constructor:

```
hash_set(System::Collections::Generic::IEnumerable<Key>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the default ordering predicate, and with the default hash function. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the default ordering predicate and hash function.

The constructor:

```
hash_set(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*, and with the default hash function. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate and default hash function.

The constructor:

```
hash_set(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred, hasher^ hashfn);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*, and with the hash function *hashfn*. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate and hash function.

## Example

```
// cliext_hash_set_construct.cpp
// compile with: /clr
#include <cliext/hash_set>

int myfun(wchar_t key)
    { // hash a key
    return (key ^ 0xdeadbeef);
    }

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
// construct an empty container
Myhash_set c1;
System::Console::WriteLine("size() = {0}", c1.size());

c1.insert(L'a');
c1.insert(L'b');
c1.insert(L'c');
for each (wchar_t elem in c1)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an ordering rule
Myhash_set c2 = cliext::greater_equal<wchar_t>();
System::Console::WriteLine("size() = {0}", c2.size());

c2.insert(c1.begin(), c1.end());
for each (wchar_t elem in c2)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an ordering rule and hash function
Myhash_set c2h(cliext::greater_equal<wchar_t>(),
    gcnew Myhash_set::hasher(&myfun));
System::Console::WriteLine("size() = {0}", c2h.size());
```

```

c2h.insert(c1.begin(), c1.end());
for each (wchar_t elem in c2h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an iterator range
Myhash_set c3(c1.begin(), c1.end());
for each (wchar_t elem in c3)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Myhash_set c4(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c4)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule and hash function
Myhash_set c4h(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_set::hasher(&myfun));
for each (wchar_t elem in c4h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an enumeration
Myhash_set c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
for each (wchar_t elem in c5)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myhash_set c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c6)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule and hash function
Myhash_set c6h( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_set::hasher(&myfun));
for each (wchar_t elem in c6h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct from a generic container
Myhash_set c7(c4);
for each (wchar_t elem in c7)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct by copying another container
Myhash_set c8(%c3);
for each (wchar_t elem in c8)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
a b c
size() = 0
a b c
size() = 0
c b a

a b c
a b c
c b a

a b c
a b c
c b a

a b c
a b c
```

## hash\_set::hasher (STL/CLR)

The hashing delegate for a key.

### Syntax

```
Microsoft::VisualC::StlClr::UnaryDelegate<GKey, int>
    hasher;
```

### Remarks

The type describes a delegate that converts a key value to an integer.

### Example

```
// cliext_hash_set_hasher.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

## hash\_set::insert (STL/CLR)

Adds elements.

### Syntax

```
cliext::pair<iterator, bool> insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);
```

## Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Key value to insert.

*where*

Where in container to insert (hint only).

## Remarks

Each of the member functions inserts a sequence specified by the remaining operands.

The first member function endeavors to insert an element with value *val*, and returns a pair of values *x*. If *x.second* is true, *x.first* designates the newly inserted element; otherwise *x.first* designates an element with equivalent ordering that already exists and no new element is inserted. You use it to insert a single element.

The second member function inserts an element with value *val*, using *where* as a hint (to improve performance), and returns an iterator that designates the newly inserted element. You use it to insert a single element which might be adjacent to an element you know.

The third member function inserts the sequence [*first*, *last*). You use it to insert zero or more elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

Each element insertion takes time proportional to the logarithm of the number of elements in the controlled sequence. Insertion can occur in amortized constant time, however, given a hint that designates an element adjacent to the insertion point.

## Example

```

// cliext_hash_set_insert.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
typedef Myhash_set::pair_iter_bool Pairib;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Pairib pair1 = c1.insert(L'x');
    System::Console::WriteLine("insert(L'x') = [{0} {1}]",
        *pair1.first, pair1.second);

    pair1 = c1.insert(L'b');
    System::Console::WriteLine("insert(L'b') = [{0} {1}]",
        *pair1.first, pair1.second);

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value with hint
    System::Console::WriteLine("insert(begin(), L'y') = {0}",
        *c1.insert(c1.begin(), L'y'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    Myhash_set c2;
    Myhash_set::iterator it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    Myhash_set c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(L'x') = [x True]
insert(L'b') = [b False]
a b c x
insert(begin(), L'y') = y
a b c x y
a b c x
a b c x y

```

## hash\_set::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

### Remarks

The type describes an object of unspecified type `T1` that can serve as a bidirectional iterator for the controlled sequence.

### Example

```
// cliext_hash_set_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myhash_set::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## hash\_set::key\_comp (STL/CLR)

Copies the ordering delegate for two keys.

### Syntax

```
key_compare^key_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two keys.

### Example

```

// cliext_hash_set_key_comp.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_Set c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## hash\_set::key\_compare (STL/CLR)

The ordering delegate for two keys.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its key arguments.

### Example

```

// cliext_hash_set_key_compare.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_Set c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## hash\_set::key\_type (STL/CLR)

The type of an ordering key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter *Key*.

### Example

```

// cliext_hash_set_key_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using key_type
    for (Myhash_Set::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myhash_Set::key_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## hash\_set::load\_factor (STL/CLR)

Counts the average elements per bucket.

### Syntax

```
float load_factor();
```

### Remarks

The member function returns `(float) hash_set::size (STL/CLR) () / hash_set::bucket_count (STL/CLR) ()`. You use it to determine the average bucket size.

### Example

```

// cliext_hash_set_load_factor.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_set::lower\_bound (STL/CLR)

Finds beginning of range that matches a specified key.

### Syntax

```
iterator lower_bound(key_type key);
```

## Parameters

*key*

Key value to search for.

## Remarks

The member function determines the first element *x* in the controlled sequence that hashes to the same bucket as *key* and has equivalent ordering to *key*. If no such element exists, it returns [hash\\_set::end \(STL/CLR\)](#) (); otherwise it returns an iterator that designates *x*. You use it to locate the beginning of a sequence of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_hash_set_lower_bound.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    System::Console::WriteLine("*lower_bound(L'a') = {0}",
        *c1.lower_bound(L'a'));
    System::Console::WriteLine("*lower_bound(L'b') = {0}",
        *c1.lower_bound(L'b')));
    return (0);
}
```

```
a b c
lower_bound(L'x') == end() = True
*lower_bound(L'a') = a
*lower_bound(L'b') = b
```

## hash\_set::make\_value (STL/CLR)

Constructs a value object.

## Syntax

```
static value_type make_value(key_type key);
```

## Parameters

*key*

Key value to use.

## Remarks

The member function returns a *value\_type* object whose key is *key*. You use it to compose an object suitable for use with several other member functions.

## Example

```
// cliext_hash_set_make_value.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(Myhash_set::make_value(L'a'));
    c1.insert(Myhash_set::make_value(L'b'));
    c1.insert(Myhash_set::make_value(L'c'));

    // display contents " a b c"
    for each (Myhash_set::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## hash\_set::max\_load\_factor (STL/CLR)

Gets or sets the maximum elements per bucket.

### Syntax

```
float max_load_factor();
void max_load_factor(float new_factor);
```

### Parameters

*new\_factor*

New maximum load factor to store.

### Remarks

The first member function returns the current stored maximum load factor. You use it to determine the maximum average bucket size.

The second member function replaces the store maximum load factor with *new\_factor*. No automatic rehashing occurs until a subsequent insert.

## Example

```

// cliext_hash_set_max_load_factor.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

## hash\_set::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
hash_set<Key>% operator=(hash_set<Key>% right);
```

### Parameters

*right*

Container to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

### Example

```

// cliext_hash_set_operator_as.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (Myhash_Set::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myhash_Set c2;
    c2 = c1;
    // display contents " a b c"
    for each (Myhash_Set::value_type elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

## hash\_set::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

### Syntax

```

reverse_iterator rbegin();

```

### Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the **beginning** of the reverse sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_hash_set_rbegin.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_Set::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("**rbegin() = {0}", *++rit);
    return (0);
}

```

```

a b c
*rbegin() = c
*++rbegin() = b

```

## hash\_set::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```

typedef value_type% reference;

```

### Remarks

The type describes a reference to an element.

### Example

```
// cliext_hash_set_reference.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myhash_Set::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        Myhash_Set::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## hash\_set::rehash (STL/CLR)

Rebuilds the hash table.

### Syntax

```
void rehash();
```

### Remarks

The member function rebuilds the hash table, ensuring that [hash\\_set::load\\_factor \(STL/CLR\)](#) ()  $\leq$  [hash\\_set::max\\_load\\_factor \(STL/CLR\)](#). Otherwise, the hash table increases in size only as needed after an insertion. (It never automatically decreases in size.) You use it to adjust the size of the hash table.

### Example

```

// cliext_hash_set_rehash.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

## hash\_set::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

## Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

## Example

```
// cliext_hash_set_rend.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_Set::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("--- --rend() = {0}", *--rit);
    System::Console::WriteLine("---rend() = {0}", *++rit);
    return (0);
}
```

```
a b c
--- --rend() = b
---rend() = a
```

## hash\_set::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

## Syntax

```
typedef T3 reverse_iterator;
```

## Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

## Example

```
// cliext_hash_set_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myhash_Set::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## hash\_set::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [hash\\_set::empty \(STL/CLR\) \(\)](#).

### Example

```

// cliext_hash_set_size.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

## hash\_set::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```

typedef int size_type;

```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_hash_set_size_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_Set::size_type diff = 0;
    for (Myhash_Set::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3

```

## hash\_set::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```

void swap(hash_set<Key>% right);

```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_hash_set_swap.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Myhash_Set c2;
    c2.insert(L'd');
    c2.insert(L'e');
    c2.insert(L'f');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
d e f
d e f
a b c

```

## hash\_set::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<value_type>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_hash_set_to_array.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c d
a b c

```

## hash\_set::upper\_bound (STL/CLR)

Finds end of range that matches a specified key.

### Syntax

```

iterator upper_bound(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the last element *x* in the controlled sequence that hashes to the same bucket as *key* and has equivalent ordering to *key*. If no such element exists, or if *x* is the last element in the controlled sequence, it returns [hash\\_set::end \(STL/CLR\) \(\)](#); otherwise it returns an iterator that designates the first element beyond *x*. You use it to locate the end of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_hash_set_upper_bound.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    System::Console::WriteLine("*upper_bound(L'a') = {0}",
        *c1.upper_bound(L'a'));
    System::Console::WriteLine("*upper_bound(L'b') = {0}",
        *c1.upper_bound(L'b'));
    return (0);
}

```

```

a b c
upper_bound(L'x') == end() = True
*upper_bound(L'a') = b
*upper_bound(L'b') = c

```

## hash\_set::value\_comp (STL/CLR)

Copies the ordering delegate for two element values.

### Syntax

```

value_compare^ value_comp();

```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two element values.

### Example

```

// cliext_hash_set_value_comp.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

```

## hash\_set::value\_compare (STL/CLR)

The ordering delegate for two element values.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its value arguments.

### Example

```

// cliext_hash_set_value_compare.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```
compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

## hash\_set::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef generic_value value_type;
```

### Remarks

The type is a synonym for `generic_value`.

### Example

```
// cliext_hash_set_value_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using value_type
    for (Myhash_Set::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Myhash_Set::value_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

# list (STL/CLR)

9/20/2022 • 44 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `list` to manage a sequence of elements as a bidirectional linked list of nodes, each storing one element.

In the description below, `GValue` is the same as `Value` unless the latter is a ref type, in which case it is `Value^`.

## Syntax

```
template<typename Value>
ref class list
    : public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    Microsoft::VisualC::StlClr::IList<GValue>
{ ..... };
```

### Parameters

*Value*

The type of an element in the controlled sequence.

## Requirements

**Header:** <cliext/list>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">list::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.
<a href="#">list::const_reference (STL/CLR)</a>	The type of a constant reference to an element.
<a href="#">list::const_reverse_iterator (STL/CLR)</a>	The type of a constant reverse iterator for the controlled sequence.
<a href="#">list::difference_type (STL/CLR)</a>	The type of a signed distance between two elements.
<a href="#">list::generic_container (STL/CLR)</a>	The type of the generic interface for the container.
<a href="#">list::generic_iterator (STL/CLR)</a>	The type of an iterator for the generic interface for the container.

TYPE DEFINITION	DESCRIPTION
<code>list::generic_reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the generic interface for the container.
<code>list::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container.
<code>list::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>list::reference (STL/CLR)</code>	The type of a reference to an element.
<code>list::reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the controlled sequence.
<code>list::size_type (STL/CLR)</code>	The type of a signed distance between two elements.
<code>list::value_type (STL/CLR)</code>	The type of an element.

MEMBER FUNCTION	DESCRIPTION
<code>list::assign (STL/CLR)</code>	Replaces all elements.
<code>list::back (STL/CLR)</code>	Accesses the last element.
<code>list::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>list::clear (STL/CLR)</code>	Removes all elements.
<code>list::empty (STL/CLR)</code>	Tests whether no elements are present.
<code>list::end (STL/CLR)</code>	Designates the end of the controlled sequence.
<code>list::erase (STL/CLR)</code>	Removes elements at specified positions.
<code>list::front (STL/CLR)</code>	Accesses the first element.
<code>list::insert (STL/CLR)</code>	Adds elements at a specified position.
<code>list::list (STL/CLR)</code>	Constructs a container object.
<code>list::merge (STL/CLR)</code>	Merges two ordered controlled sequences.
<code>list::pop_back (STL/CLR)</code>	Removes the last element.
<code>list::pop_front (STL/CLR)</code>	Removes the first element.
<code>list::push_back (STL/CLR)</code>	Adds a new last element.
<code>list::push_front (STL/CLR)</code>	Adds a new first element.
<code>list::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.

MEMBER FUNCTION	DESCRIPTION
<code>list::remove (STL/CLR)</code>	Removes an element with a specified value.
<code>list::remove_if (STL/CLR)</code>	Removes elements that pass a specified test.
<code>list::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>list::resize (STL/CLR)</code>	Changes the number of elements.
<code>list::reverse (STL/CLR)</code>	Reverses the controlled sequence.
<code>list::size (STL/CLR)</code>	Counts the number of elements.
<code>list::sort (STL/CLR)</code>	Orders the controlled sequence.
<code>list::splice (STL/CLR)</code>	Restitches links between nodes.
<code>list::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>list::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>list::unique (STL/CLR)</code>	Removes adjacent elements that pass a specified test.

PROPERTY	DESCRIPTION
<code>list::back_item (STL/CLR)</code>	Accesses the last element.
<code>list::front_item (STL/CLR)</code>	Accesses the first element.

  

OPERATOR	DESCRIPTION
<code>list::operator= (STL/CLR)</code>	Replaces the controlled sequence.
<code>operator!= (list) (STL/CLR)</code>	Determines if a <code>list</code> object is not equal to another <code>list</code> object.
<code>operator&lt; (list) (STL/CLR)</code>	Determines if a <code>list</code> object is less than another <code>list</code> object.
<code>operator&lt;= (list) (STL/CLR)</code>	Determines if a <code>list</code> object is less than or equal to another <code>list</code> object.
<code>operator== (list) (STL/CLR)</code>	Determines if a <code>list</code> object is equal to another <code>list</code> object.
<code>operator&gt; (list) (STL/CLR)</code>	Determines if a <code>list</code> object is greater than another <code>list</code> object.
<code>operator&gt;= (list) (STL/CLR)</code>	Determines if a <code>list</code> object is greater than or equal to another <code>list</code> object.

# Interfaces

INTERFACE	DESCRIPTION
<a href="#">ICloneable</a>	Duplicate an object.
<a href="#">IEnumerable</a>	Sequence through elements.
<a href="#">ICollection</a>	Maintain group of elements.
<a href="#">IEnumerable&lt;T&gt;</a>	Sequence through typed elements.
<a href="#">ICollection&lt;T&gt;</a>	Maintain group of typed elements.
<a href="#">IList&lt;Value&gt;</a>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes in a bidirectional link list. It rearranges elements by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements. Thus, a list is a good candidate for the underlying container for template class [queue \(STL/CLR\)](#) or template class [stack \(STL/CLR\)](#).

A `list` object supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by `list::end (STL/CLR)()`. You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a list iterator to reach the head node, and it will then compare equal to `end()`. But you cannot dereference the iterator returned by `end()`.

Note that you cannot refer to a list element directly given its numerical position -- that requires a random-access iterator. So a list is *not* usable as the underlying container for template class [priority\\_queue \(STL/CLR\)](#).

A list iterator stores a handle to its associated list node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A list iterator remains valid so long as its associated list node is associated with some list. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### `list::assign (STL/CLR)`

Replaces all elements.

#### Syntax

```
void assign(size_type count, value_type val);
template<typename InIt>
    void assign(InIt first, InIt last);
void assign(System::Collections::Generic::IEnumerable<Value>^ right);
```

## Parameters

*count*

Number of elements to insert.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Value of the element to insert.

## Remarks

The first member function replaces the controlled sequence with a repetition of *count* elements of value *val*. You use it to fill the container with elements all having the same value.

If *InIt* is an integer type, the second member function behaves the same as

`assign((size_type)first, (value_type)last)`. Otherwise, it replaces the controlled sequence with the sequence [*first*, *last*). You use it to make the controlled sequence a copy another sequence.

The third member function replaces the controlled sequence with the sequence designated by the enumerator *right*. You use it to make the controlled sequence a copy of a sequence described by an enumerator.

## Example

```
// cliext_list_assign.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // assign a repetition of values
    cliext::list<wchar_t> c2;
    c2.assign(6, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an iterator range
    cliext::list<wchar_t>::iterator it = c1.end();
    c2.assign(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an enumeration
    c2.assign( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
x x x x x x
a b
a b c
```

## list::back (STL/CLR)

Accesses the last element.

### Syntax

```
reference back();
```

### Remarks

The member function returns a reference to the last element of the controlled sequence, which must be non-empty. You use it to access the last element, when you know it exists.

### Example

```
// cliext_list_back.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back() = {0}", c1.back());

    // alter last item and reinspect
    c1.back() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
back() = c
a b x
```

## list::back\_item (STL/CLR)

Accesses the last element.

### Syntax

```
property value_type back_item;
```

## Remarks

The property accesses the last element of the controlled sequence, which must be non-empty. You use it to read or write the last element, when you know it exists.

## Example

```
// cliext_list_back_item.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back_item = {0}", c1.back_item);

    // alter last item and reinspect
    c1.back_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
back_item = c
a b x
```

## list::begin (STL/CLR)

Designates the beginning of the controlled sequence.

### Syntax

```
iterator begin();
```

## Remarks

The member function returns a random-access iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

## Example

```

// cliext_list_begin.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::list<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("++begin() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*begin() = a
++begin() = b
x y c

```

## list::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

### Remarks

The member function effectively calls `list::erase (STL/CLR) ( list::begin (STL/CLR) (), list::end (STL/CLR) () )`. You use it to ensure that the controlled sequence is empty.

### Example

```

// cliext_list_clear.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

a b c
size() = 0
a b
size() = 0

```

## list::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

### Syntax

```

typedef T2 const_iterator;

```

### Remarks

The type describes an object of unspecified type `T2` that can serve as a constant random-access iterator for the controlled sequence.

### Example

```

// cliext_list_const_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::list<wchar_t>::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## list::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_list_const_reference.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::list<wchar_t>::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        { // get a const reference to an element
        cliext::list<wchar_t>::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## list::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```
// cliext_list_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::list<wchar_t>::const_reverse_iterator crit = c1.rbegin();
    cliext::list<wchar_t>::const_reverse_iterator crend = c1.rend();
    for (; crit != crend; ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## list::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a signed element count.

### Example

```

// cliext_list_difference_type.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::list<wchar_t>::difference_type diff = 0;
    for (cliext::list<wchar_t>::iterator it = c1.begin();
         it != c1.end(); ++it) ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (cliext::list<wchar_t>::iterator it = c1.end();
         it != c1.begin(); --it) --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

## list::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

### Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [list::size \(STL/CLR\)](#)  
`() == 0`. You use it to test whether the list is empty.

### Example

```

// cliext_list_empty.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

## list::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a random-access iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_list_end.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    cliext::list<wchar_t>::iterator it = c1.end();
    --it;
    System::Console::WriteLine("---- --end() = {0}", *--it);
    System::Console::WriteLine("----end() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
---- --end() = b
----end() = c
a x y

```

## list::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);

```

### Parameters

*first*

Beginning of range to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [`first`, `last`]. You

use it to remove zero or more contiguous elements.

Both member functions return an iterator that designates the first element remaining beyond any elements removed, or [list::end \(STL/CLR\)](#) if no such element exists.

When erasing elements, the number of element copies is linear in the number of elements between the end of the erasure and the nearer end of the sequence. (When erasing one or more elements at either end of the sequence, no element copies occur.)

## Example

```
// cliext_list_erase.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.push_back(L'd');
    c1.push_back(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    cliext::list<wchar_t>::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

## list::front (STL/CLR)

Accesses the first element.

### Syntax

```
reference front();
```

### Remarks

The member function returns a reference to the first element of the controlled sequence, which must be non-empty. You use it to read or write the first element, when you know it exists.

### Example

```
// cliext_list_front.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front() = {0}", c1.front());

    // alter first item and reinspect
    c1.front() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front() = a
x b c
```

## list::front\_item (STL/CLR)

Accesses the first element.

### Syntax

```
property value_type front_item;
```

### Remarks

The property accesses the first element of the controlled sequence, which must be non-empty. You use it to read or write the first element, when you know it exists.

### Example

```

// cliext_list_front_item.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front_item = {0}", c1.front_item);

    // alter first item and reinspect
    c1.front_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
front_item = a
x b c

```

## list::generic\_container (STL/CLR)

The type of the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::
    IList<generic_value>
generic_container;

```

### Remarks

The type describes the generic interface for this template container class.

### Example

```

// cliext_list_generic_container.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::list<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(gc1->end(), L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push_back(L'e');

    System::Collections::IEnumerator^ enum1 =
        gc1->GetEnumerator();
    while (enum1->MoveNext())
        System::Console::Write("{0} ", enum1->Current);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

## list::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;

```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_list_generic_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::list<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::list<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::list<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

## list::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseBidirectionalIterator<generic_value> generic_reverse_iterator;

```

### Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_list_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::list<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::list<wchar_t>::generic_reverse_iterator gcit = gc1->rbegin();
    cliext::list<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a c c

```

## list::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```
typedef GValue generic_value;
```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

### Example

```

// cliext_list_generic_value.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::list<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::list<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::list<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

## list::insert (STL/CLR)

Adds elements at a specified position.

### Syntax

```

iterator insert(iterator where, value_type val);
void insert(iterator where, size_type count, value_type val);
template<typename InIt>
    void insert(iterator where, InIt first, InIt last);
void insert(iterator where,
    System::Collections::Generic::IEnumerable<Value>^ right);

```

### Parameters

*count*

Number of elements to insert.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Value of the element to insert.

*where*

Where in container to insert before.

## Remarks

Each of the member functions inserts, before the element pointed to by *where* in the controlled sequence, a sequence specified by the remaining operands.

The first member function inserts an element with value *val* and returns an iterator that designates the newly inserted element. You use it to insert a single element before a place designated by an iterator.

The second member function inserts a repetition of *count* elements of value *val*. You use it to insert zero or more contiguous elements which are all copies of the same value.

If `Init` is an integer type, the third member function behaves the same as

`insert(where, (size_type)first, (value_type)last)`. Otherwise, it inserts the sequence `[ first, last ]`. You use it to insert zero or more contiguous elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

When inserting a single element, the number of element copies is linear in the number of elements between the insertion point and the nearer end of the sequence. (When inserting one or more elements at either end of the sequence, no element copies occur.) If `Init` is an input iterator, the third member function effectively performs a single insertion for each element in the sequence. Otherwise, when inserting `N` elements, the number of element copies is linear in `N` plus the number of elements between the insertion point and the nearer end of the sequence.

## Example

```

// cliext_list_insert.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value using iterator
    cliext::list<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("insert(begin())+1, L'x') = {0}",
        *c1.insert(++it, L'x'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a repetition of values
    cliext::list<wchar_t> c2;
    c2.insert(c2.begin(), 2, L'y');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    it = c1.end();
    c2.insert(c2.end(), c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    c2.insert(c2.begin(), // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value using index
    it = c2.begin();
    ++it, ++it, ++it;
    c2.insert(it, L'z');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}

```

```

a b c
insert(begin())+1, L'x') = x
a x b c
y y
y y a x b
a x b c y y a x b

```

## list::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

### Remarks

The type describes an object of unspecified type `T1` that can serve as a random-access iterator for the controlled sequence.

### Example

```
// cliext_list_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::list<wchar_t>::iterator it = c1.begin();
    for ( ; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();

    // alter first element and redisplay
    it = c1.begin();
    *it = L'x';
    for ( ; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x b c
```

## list::list (STL/CLR)

Constructs a container object.

### Syntax

```
list();
list(list<Value>% right);
list(list<Value>^ right);
explicit list(size_type count);
list(size_type count, value_type val);
template<typename InIt>
    list(InIt first, InIt last);
list(System::Collections::Generic::IEnumarable<Value>^ right);
```

### Parameters

*count*

Number of elements to insert.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Object or range to insert.

*val*

Value of the element to insert.

## Remarks

The constructor:

```
list();
```

initializes the controlled sequence with no elements. You use it to specify an empty initial controlled sequence.

The constructor:

```
list(list<Value>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`]. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the list object *right*.

The constructor:

```
list(list<Value>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`]. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the list object whose handle is *right*.

The constructor:

```
explicit list(size_type count);
```

initializes the controlled sequence with *count* elements each with value `value_type()`. You use it to fill the container with elements all having the default value.

The constructor:

```
list(size_type count, value_type val);
```

initializes the controlled sequence with *count* elements each with value *val*. You use it to fill the container with elements all having the same value.

The constructor:

```
template<typename InIt>
```

```
list(InIt first, InIt last);
```

initializes the controlled sequence with the sequence [`first`, `last`]. You use it to make the controlled sequence a copy of another sequence.

The constructor:

```
list(System::Collections::Generic::IEnumerable<Value>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*. You use it to make the controlled sequence a copy of another sequence described by an enumerator.

## Example

```
// cliext_list_construct.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
// construct an empty container
    cliext::list<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());

// construct with a repetition of default values
    cliext::list<wchar_t> c2(3);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

// construct with a repetition of values
    cliext::list<wchar_t> c3(6, L'x');
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range
    cliext::list<wchar_t>::iterator it = c3.end();
    cliext::list<wchar_t> c4(c3.begin(), --it);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an enumeration
    cliext::list<wchar_t> c5( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
    for each (wchar_t elem in c5)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying another container
    cliext::list<wchar_t> c7(c3);
    for each (wchar_t elem in c7)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying a container handle
    cliext::list<wchar_t> c8(%c3);
    for each (wchar_t elem in c8)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}
```

```
size() = 0
0 0 0
x x x x x x
x x x x x
x x x x x x
x x x x x x
x x x x x x
```

## list::merge (STL/CLR)

Merges two ordered controlled sequences.

### Syntax

```
void merge(list<Value>% right);
template<typename Pred2>
    void merge(list<Value>% right, Pred2 pred);
```

### Parameters

*pred*

Comparer for element pairs.

*right*

Container to merge in.

### Remarks

The first member function removes all elements from the sequence controlled by *right* and insert them in the controlled sequence. Both sequences must be previously ordered by `operator<` -- elements must not decrease in value as you progress through either sequence. The resulting sequence is also ordered by `operator<`. You use this member function to merge two sequences that increase in value into a sequence that also increases in value.

The second member function behaves the same as the first, except that the sequences are ordered by `pred` -- `pred(x, y)` must be false for any element `x` that follows element `y` in the sequence. You use it to merge two sequences ordered by a predicate function or delegate that you specify.

Both functions perform a stable merge -- no pair of elements in either of the original controlled sequences is reversed in the resulting controlled sequence. Also, if a pair of elements `x` and `y` in the resulting controlled sequence has equivalent ordering -- `!(x < y) && !(y < x)` -- an element from the original controlled sequence appears before an element from the sequence controlled by *right*.

### Example

```

// cliext_list_merge.cpp
// compile with: /clr
#include <cliext/list>

typedef cliext::list<wchar_t> Mylist;
int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'c');
    c1.push_back(L'e');

    // display initial contents " a c e"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    cliext::list<wchar_t> c2;
    c2.push_back(L'b');
    c2.push_back(L'd');
    c2.push_back(L'f');

    // display initial contents " b d f"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // merge and display
    cliext::list<wchar_t> c3(c1);
    c3.merge(c2);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("c2.size() = {0}", c2.size());

    // sort descending, merge descending, and redisplay
    c1.sort(cliext::greater<wchar_t>());
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    c3.sort(cliext::greater<wchar_t>());
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    c3.merge(c1, cliext::greater<wchar_t>());
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("c1.size() = {0}", c1.size());
    return (0);
}

```

```

a c e
b d f
a b c d e f
c2.size() = 0
e c a
f e d c b a
f e e d c c b a a
c1.size() = 0

```

## list::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
list<Value>% operator=(list<Value>% right);
```

### Parameters

*right*

Container to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

### Example

```
// cliext_list_operator_as.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

## list::pop\_back (STL/CLR)

Removes the last element.

### Syntax

```
void pop_back();
```

### Remarks

The member function removes the last element of the controlled sequence, which must be non-empty. You use it to shorten the list by one element at the back.

## Example

```
// cliext_list_pop_back.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_back();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b
```

## list::pop\_front (STL/CLR)

Removes the first element.

### Syntax

```
void pop_front();
```

### Remarks

The member function removes the first element of the controlled sequence, which must be non-empty. You use it to shorten the list by one element at the front.

## Example

```

// cliext_list_pop_front.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_front();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
b c

```

## list::push\_back (STL/CLR)

Adds a new last element.

### Syntax

```

void push_back(value_type val);

```

### Remarks

The member function inserts an element with value `val` at the end of the controlled sequence. You use it to append another element to the list.

### Example

```

// cliext_list_push_back.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

## list::push\_front (STL/CLR)

Adds a new first element.

### Syntax

```
void push_front(value_type val);
```

### Remarks

The member function inserts an element with value `val` at the beginning of the controlled sequence. You use it to prepend another element to the list.

### Example

```
// cliext_list_push_front.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_front(L'a');
    c1.push_front(L'b');
    c1.push_front(L'c');

    // display contents " c b a"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## list::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

### Syntax

```
reverse_iterator rbegin();
```

### Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the `beginning` of the reverse sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_list_rbegin.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    cliext::list<wchar_t>::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("++rbegin() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*rbegin() = c
++rbegin() = b
a y x

```

## list::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```

typedef value_type% reference;

```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_list_reference.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::list<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        cliext::list<wchar_t>::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();

    // modify contents " a b c"
    for (it = c1.begin(); it != c1.end(); ++it)
        { // get a reference to an element
        cliext::list<wchar_t>::reference ref = *it;

        ref += (wchar_t)(L'A' - L'a');
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
A B C

```

## list::remove (STL/CLR)

Removes an element with a specified value.

### Syntax

```

void remove(value_type val);

```

### Parameters

*val*

Value of the element to remove.

### Remarks

The member function removes an element in the controlled sequence for which

`((System::Object^)val)->Equals((System::Object^)x)` is true (if any). You use it to erase an arbitrary element with the specified value.

### Example

```

// cliext_list_remove.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // fail to remove and redisplay
    c1.remove(L'A');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // remove and redisplay
    c1.remove(L'b');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a c

```

## list::remove\_if (STL/CLR)

Removes elements that pass a specified test.

### Syntax

```

template<typename Pred1>
void remove_if(Pred1 pred);

```

### Parameters

*pred*

Test for elements to remove.

### Remarks

The member function removes from the controlled sequence (erases) every element *x* for which *pred(x)* is true. You use it to remove all elements that satisfy a condition you specify as a function or delegate.

### Example

```

// cliext_list_remove_if.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'b');
    c1.push_back(L'b');
    c1.push_back(L'b');

    // display initial contents " a b b b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // fail to remove and redisplay
    c1.remove_if(cliext::binder2nd<cliext::equal_to<wchar_t> >(
        cliext::equal_to<wchar_t>(), L'd'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // remove and redisplay
    c1.remove_if(cliext::binder2nd<cliext::not_equal_to<wchar_t> >(
        cliext::not_equal_to<wchar_t>(), L'b'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b b b c
a b b b c
b b b

```

## list::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

### Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_list_rend.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::list<wchar_t>::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("---- --rend() = {0}", *--rit);
    System::Console::WriteLine("----rend() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
---- --rend() = b
----rend() = a
y x c

```

## list::resize (STL/CLR)

Changes the number of elements.

### Syntax

```

void resize(size_type new_size);
void resize(size_type new_size, value_type val);

```

### Parameters

*new\_size*

New size of the controlled sequence.

*val*

Value of the padding element.

### Remarks

The member functions both ensure that [list::size \(STL/CLR\) \(\)](#) henceforth returns *new\_size*. If it must make the controlled sequence longer, the first member function appends elements with value [value\\_type\(\)](#), while the second member function appends elements with value *val*. To make the controlled sequence shorter, both member functions effectively erase the last element [list::size \(STL/CLR\) \(\) - new\\_size](#) times. You use it to ensure that the controlled sequence has size *new\_size*, by either trimming or padding the current controlled sequence.

## Example

```
// cliext_list_resize.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
// construct an empty container and pad with default values
    cliext::list<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());
    c1.resize(4);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

    // resize to empty
    c1.resize(0);
    System::Console::WriteLine("size() = {0}", c1.size());

    // resize and pad
    c1.resize(5, L'x');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
size() = 0
0 0 0 0
size() = 0
x x x x x
```

## list::reverse (STL/CLR)

Reverses the controlled sequence.

### Syntax

```
void reverse();
```

### Remarks

The member function reverses the order of all elements in the controlled sequence. You use it to reflect a list of elements.

## Example

```
// cliext_list_reverse.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // reverse and redisplay
    c1.reverse();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
c b a
```

## list::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

### Syntax

```
typedef T3 reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

### Example

```

// cliext_list_reverse_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::list<wchar_t>::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();

    // alter first element and redisplay
    rit = c1.rbegin();
    *rit = L'x';
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}

```

```

c b a
x b a

```

## list::size (STL/CLR)

Counts the number of elements.

### Syntax

```

size_type size();

```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [list::empty \(STL/CLR\)](#).

### Example

```

// cliext_list_size.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

## list::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_list_size_type.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::list<wchar_t>::size_type diff = 0;
    for (cliext::list<wchar_t>::iterator it = c1.begin();
         it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3

```

## list::sort (STL/CLR)

Orders the controlled sequence.

### Syntax

```

void sort();
template<typename Pred2>
void sort(Pred2 pred);

```

### Parameters

*pred*

Comparer for element pairs.

### Remarks

The first member function rearranges the elements in the controlled sequence so that they are ordered by `operator<` -- elements do not decrease in value as you progress through the sequence. You use this member function to sort the sequence in increasing order.

The second member function behaves the same as the first, except that the sequence is ordered by `pred` -- `pred(x, y)` is false for any element `x` that follows element `y` in the resultant sequence. You use it to sort the sequence in an order that you specify by a predicate function or delegate.

Both functions perform a stable sort -- no pair of elements in the original controlled sequence is reversed in the resulting controlled sequence.

### Example

```

// cliext_list_sort.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // sort descending and redisplay
    c1.sort(cliext::greater<wchar_t>());
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // sort ascending and redisplay
    c1.sort();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
c b a
a b c

```

## list::splice (STL/CLR)

Restitch links between nodes.

### Syntax

```

void splice(iterator where, list<Value>% right);
void splice(iterator where, list<Value>% right,
            iterator first);
void splice(iterator where, list<Value>% right,
            iterator first, iterator last);

```

### Parameters

*first*

Beginning of range to splice.

*last*

End of range to splice.

*right*

Container to splice from.

*where*

Where in container to splice before.

### Remarks

The first member function inserts the sequence controlled by *right* before the element in the controlled sequence pointed to by *where*. It also removes all elements from *right*. (`%right` must not equal `this`.) You use it to splice all of one list into another.

The second member function removes the element pointed to by *first* in the sequence controlled by *right* and inserts it before the element in the controlled sequence pointed to by *where*. (If `where == first` || `where == ++first`, no change occurs.) You use it to splice a single element of one list into another.

The third member function inserts the subrange designated by `[first, last)` from the sequence controlled by *right* before the element in the controlled sequence pointed to by *where*. It also removes the original subrange from the sequence controlled by *right*. (If `right == this`, the range `[first, last)` must not include the element pointed to by *where*.) You use it to splice a subsequence of zero or more elements from one list into another.

## Example

```
// cliext_list_splice.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // splice to a new list
    cliext::list<wchar_t> c2;
    c2.splice(c2.begin(), c1);
    System::Console::WriteLine("c1.size() = {0}", c1.size());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // return one element
    c1.splice(c1.end(), c2, c2.begin());
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // return remaining elements
    c1.splice(c1.begin(), c2, c2.begin(), c2.end());
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("c2.size() = {0}", c2.size());
    return (0);
}
```

```
a b c
c1.size() = 0
a b c
a
b c
b c a
c2.size() = 0
```

## list::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```
void swap(list<Value>% right);
```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `*this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```
// cliext_list_swap.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::list<wchar_t> c2(5, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x x x x x
x x x x x
a b c
```

## list::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```
cli::array<Value>^ to_array();
```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```
// cliext_list_to_array.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push_back(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c d
a b c
```

## list::unique (STL/CLR)

Removes adjacent elements that pass a specified test.

### Syntax

```
void unique();
template<typename Pred2>
    void unique(Pred2 pred);
```

## Parameters

*pred*

Comparer for element pairs.

## Remarks

The first member function removes from the controlled sequence (erases) every element that compares equal to its preceding element -- if element *x* precedes element *y* and *x == y*, the member function removes *y*. You use it to remove all but one copy of every subsequence of adjacent elements that compare equal. Note that if the controlled sequence is ordered, such as by calling [list::sort \(STL/CLR\)](#) (), the member function leaves only elements with unique values. (Hence the name).

The second member function behaves the same as the first, except that it removes each element *y* following an element *x* for which *pred(x, y)*. You use it to remove all but one copy of every subsequence of adjacent elements that satisfy a predicate function or delegate that you specify. Note that if the controlled sequence is ordered, such as by calling [sort\(pred\)](#), the member function leaves only elements that do not have equivalent ordering with any other elements.

## Example

```
// cliext_list_unique.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display contents after unique
    cliext::list<wchar_t> c2(c1);
    c2.unique();
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display contents after unique(not_equal_to)
    c2 = c1;
    c2.unique(cliext::not_equal_to<wchar_t>());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a a b c
a b c
a a
```

## list::value\_type (STL/CLR)

The type of an element.

## Syntax

```
typedef Value value_type;
```

## Remarks

The type is a synonym for the template parameter *Value*.

## Example

```
// cliext_list_value_type.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using value_type
    for (cliext::list<wchar_t>::iterator it = c1.begin();
        it != c1.end(); ++it)
    {
        // store element in value_type object
        cliext::list<wchar_t>::value_type val = *it;

        System::Console::Write("{0} ", val);
    }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## operator!= (list) (STL/CLR)

List not equal comparison.

## Syntax

```
template<typename Value>
bool operator!=(list<Value>% left,
                 list<Value>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two lists are compared element by element.

## Example

```

// cliext_list_operator_ne.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

## operator< (list) (STL/CLR)

List less than comparison.

### Syntax

```

template<typename Value>
bool operator<(list<Value>% left,
                 list<Value>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true if, for the lowest position *i* for which  $\text{!(right[i] < left[i])}$  it is also true that  $\text{left[i] < right[i]}$ . Otherwise, it returns  $\text{left->size() < right->size()}$ . You use it to test whether *left* is ordered before *right* when the two lists are compared element by element.

## Example

```
// cliext_list_operator_lt.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

## operator<= (list) (STL/CLR)

List less than or equal comparison.

### Syntax

```
template<typename Value>
bool operator<=(list<Value>% left,
    list<Value>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two lists are compared element by element.

## Example

```
// cliext_list_operator_le.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator== (list) (STL/CLR)

List equal comparison.

### Syntax

```
template<typename Value>
bool operator==(list<Value>% left,
                 list<Value>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position *i*, *left[i] == right[i]*. You use it to test whether *left* is ordered the same as *right* when the two lists are compared element by element.

## Example

```
// cliext_list_operator_eq.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

## operator> (list) (STL/CLR)

List greater than comparison.

## Syntax

```
template<typename Value>
bool operator>(list<Value>% left,
    list<Value>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two lists are compared element by element.

## Example

```
// cliext_list_operator_gt.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

## operator>= (list) (STL/CLR)

List greater than or equal comparison.

## Syntax

```
template<typename Value>
bool operator>=(list<Value>% left,
    list<Value>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two lists are compared element by element.

## Example

```
// cliext_list_operator_ge.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
        c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

# map (STL/CLR)

9/20/2022 • 43 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `map` to manage a sequence of elements as a (nearly) balanced ordered tree of nodes, each storing one element. An element consists of a key, for ordering the sequence, and a mapped value, which goes along for the ride.

In the description below, `GValue` is the same as:

`Microsoft::VisualC::StlClr::GenericPair<GKey, GMapped>`

where:

`GKey` is the same as *Key* unless the latter is a ref type, in which case it is `Key^`

`GMapped` is the same as *Mapped* unless the latter is a ref type, in which case it is `Mapped^`

## Syntax

```
template<typename Key,
         typename Mapped>
ref class map
{
    public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    System::Collections::Generic::IDictionary<Gkey, GMapped>,
    Microsoft::VisualC::StlClr::ITree<Gkey, GValue>
{ .....
```

### Parameters

#### *Key*

The type of the key component of an element in the controlled sequence.

#### *Mapped*

The type of the additional component of an element in the controlled sequence.

## Requirements

**Header:** <cliext/map>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">map::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.

TYPE DEFINITION	DESCRIPTION
<code>map::const_reference (STL/CLR)</code>	The type of a constant reference to an element.
<code>map::const_reverse_iterator (STL/CLR)</code>	The type of a constant reverse iterator for the controlled sequence.
<code>map::difference_type (STL/CLR)</code>	The type of a (possibly signed) distance between two elements.
<code>map::generic_container (STL/CLR)</code>	The type of the generic interface for the container.
<code>map::generic_iterator (STL/CLR)</code>	The type of an iterator for the generic interface for the container.
<code>map::generic_reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the generic interface for the container.
<code>map::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container.
<code>map::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>map::key_compare (STL/CLR)</code>	The ordering delegate for two keys.
<code>map::key_type (STL/CLR)</code>	The type of an ordering key.
<code>map::mapped_type (STL/CLR)</code>	The type of the mapped value associated with each key.
<code>map::reference (STL/CLR)</code>	The type of a reference to an element.
<code>map::reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the controlled sequence.
<code>map::size_type (STL/CLR)</code>	The type of a (non-negative) distance between two elements.
<code>map::value_compare (STL/CLR)</code>	The ordering delegate for two element values.
<code>map::value_type (STL/CLR)</code>	The type of an element.
MEMBER FUNCTION	DESCRIPTION
<code>map::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>map::clear (STL/CLR)</code>	Removes all elements.
<code>map::count (STL/CLR)</code>	Counts elements matching a specified key.
<code>map::empty (STL/CLR)</code>	Tests whether no elements are present.
<code>map::end (STL/CLR)</code>	Designates the end of the controlled sequence.
<code>map::equal_range (STL/CLR)</code>	Finds range that matches a specified key.

MEMBER FUNCTION	DESCRIPTION
<code>map::erase (STL/CLR)</code>	Removes elements at specified positions.
<code>map::find (STL/CLR)</code>	Finds an element that matches a specified key.
<code>map::insert (STL/CLR)</code>	Adds elements.
<code>map::key_comp (STL/CLR)</code>	Copies the ordering delegate for two keys.
<code>map::lower_bound (STL/CLR)</code>	Finds beginning of range that matches a specified key.
<code>map::make_value (STL/CLR)</code>	Constructs a value object.
<code>map::map (STL/CLR)</code>	Constructs a container object.
<code>map::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.
<code>map::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>map::size (STL/CLR)</code>	Counts the number of elements.
<code>map::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>map::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>map::upper_bound (STL/CLR)</code>	Finds end of range that matches a specified key.
<code>map::value_comp (STL/CLR)</code>	Copies the ordering delegate for two element values.

OPERATOR	DESCRIPTION
<code>map::operator= (STL/CLR)</code>	Replaces the controlled sequence.
<code>map::operator[](STL/CLR)</code>	Maps a key to its associated mapped value.
<code>operator!= (map) (STL/CLR)</code>	Determines if a <code>map</code> object is not equal to another <code>map</code> object.
<code>operator&lt; (map) (STL/CLR)</code>	Determines if a <code>map</code> object is less than another <code>map</code> object.
<code>operator&lt;= (map) (STL/CLR)</code>	Determines if a <code>map</code> object is less than or equal to another <code>map</code> object.
<code>operator== (map) (STL/CLR)</code>	Determines if a <code>map</code> object is equal to another <code>map</code> object.
<code>operator&gt; (map) (STL/CLR)</code>	Determines if a <code>map</code> object is greater than another <code>map</code> object.

OPERATOR	DESCRIPTION
<code>operator&gt;= (map) (STL/CLR)</code>	Determines if a <code>map</code> object is greater than or equal to another <code>map</code> object.

## Interfaces

INTERFACE	DESCRIPTION
<code>ICloneable</code>	Duplicate an object.
<code>IEnumerable</code>	Sequence through elements.
<code>ICollection</code>	Maintain group of elements.
<code>IEnumerable&lt;T&gt;</code>	Sequence through typed elements.
<code>ICollection&lt;T&gt;</code>	Maintain group of typed elements.
<code>IDictionary&lt; TKey, TValue &gt;</code>	Maintain group of {key, value} pairs.
<code>ITree&lt;Key, Value&gt;</code>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes. It inserts elements into a (nearly) balanced tree that it keeps ordered by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements.

The object orders the sequence it controls by calling a stored delegate object of type `map::key_compare (STL/CLR)`. You can specify the stored delegate object when you construct the map; if you specify no delegate object, the default is the comparison `operator<(key_type, key_type)`. You access this stored object by calling the member function `map::key_comp (STL/CLR) ()`.

Such a delegate object must impose a strict weak ordering on keys of type `map::key_type (STL/CLR)`. That means, for any two keys `x` and `y`:

`key_comp()(x, y)` returns the same Boolean result on every call.

If `key_comp()(x, y)` is true, then `key_comp()(y, x)` must be false.

If `key_comp()(x, y)` is true, then `x` is said to be ordered before `y`.

If `!key_comp()(x, y) && !key_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

For any element `x` that precedes `y` in the controlled sequence, `key_comp()(y, x)` is false. (For the default delegate object, keys never decrease in value.) Unlike template class `map`, an object of template class `map` does not require that keys for all elements are unique. (Two or more keys can have equivalent ordering.)

Each element contains a separate key and a mapped value. The sequence is represented in a way that permits lookup, insertion, and removal of an arbitrary element with a number of operations proportional to the logarithm of the number of elements in the sequence (logarithmic time). Moreover, inserting an element invalidates no iterators, and removing an element invalidates only those iterators which point at the removed

element.

A map supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by `map::end (STL/CLR)`. You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a map iterator to reach the head node, and it will then compare equal to `end()`. But you cannot dereference the iterator returned by `end()`.

Note that you cannot refer to a map element directly given its numerical position -- that requires a random-access iterator.

A map iterator stores a handle to its associated map node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A map iterator remains valid so long as its associated map node is associated with some map. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### map::begin (STL/CLR)

Designates the beginning of the controlled sequence.

#### Syntax

```
iterator begin();
```

#### Remarks

The member function returns a bidirectional iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

#### Example

```

// cliext_map_begin.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items
    Mymap::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*++begin() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*begin() = [a 1]
*++begin() = [b 2]

```

## map::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

### Remarks

The member function effectively calls `map::erase (STL/CLR) ( map::begin (STL/CLR) (), map::end (STL/CLR) () )`. You use it to ensure that the controlled sequence is empty.

### Example

```

// cliext_map_clear.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));

    // display contents " [a 1] [b 2]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2]
size() = 0

```

## map::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

### Syntax

```
typedef T2 const_iterator;
```

### Remarks

The type describes an object of unspecified type `T2` that can serve as a constant bidirectional iterator for the controlled sequence.

### Example

```

// cliext_map_const_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("[{0} {1}] ", cit->first, cit->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## map::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_map_const_reference.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Mymap::const_reference cref = *cit;
        System::Console::Write("[{0} {1}] ", cref->first, cref->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
```

## map::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```
// cliext_map_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Mymap::const_reverse_iterator crit = c1.rbegin();
    for ( ; crit != c1.rend(); ++crit)
        System::Console::Write("[{0} {1}] ", crit->first, crit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

## map::count (STL/CLR)

Finds the number of elements matching a specified key.

### Syntax

```
size_type count(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns the number of elements in the controlled sequence that have equivalent ordering with *key*. You use it to determine the number of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_map_count.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

## map::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a possibly negative element count.

## Example

```

// cliext_map_difference_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Mymap::difference_type diff = 0;
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Mymap::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3
begin()-end() = -3

```

## map::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

### Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [map::size \(STL/CLR\) \(\) == 0](#). You use it to test whether the map is empty.

### Example

```

// cliext_map_empty.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 3
empty() = False
size() = 0
empty() = True

```

## map::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a bidirectional iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_map_end.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect last two items
    Mymap::iterator it = c1.end();
    --it;
    --it;
    System::Console::WriteLine("---- --end() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("----end() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

## map::equal\_range (STL/CLR)

Finds range that matches a specified key.

### Syntax

```
cliext::pair<iterator, iterator> equal_range(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns a pair of iterators `cliext::pair<iterator, iterator>()` (`map::lower_bound (STL/CLR) (key)`, `map::upper_bound (STL/CLR) (key)`). You use it to determine the range of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_map_equal_range.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
typedef Mymap::pair_iter_pair Pairii;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("[{0} {1}] ",
            pair1.first->first, pair1.first->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
equal_range(L'x') empty = True
[b 2]

```

## map::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

### Parameters

*first*

Beginning of range to erase.

*key*

Key value to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*, and returns an iterator that designates the first element remaining beyond the element removed, or [map::end \(STL/CLR\)](#) if no such element exists. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [ *first* , *last* ), and returns an iterator that designates the first element remaining beyond any elements removed, or *end()* if no such element exists.. You use it to remove zero or more contiguous elements.

The third member function removes any element of the controlled sequence whose key has equivalent ordering to *key*, and returns a count of the number of elements removed. You use it to remove and count all elements that match a specified key.

Each element erasure takes time proportional to the logarithm of the number of elements in the controlled sequence.

## Example

```
// cliext_map_erase.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    cliext::map<wchar_t, int> c1;
    c1.insert(cliext::map<wchar_t, int>::make_value(L'a', 1));
    c1.insert(cliext::map<wchar_t, int>::make_value(L'b', 2));
    c1.insert(cliext::map<wchar_t, int>::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (cliext::map<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase an element and reinspect
    cliext::map<wchar_t, int>::iterator it =
        c1.erase(c1.begin());
    System::Console::WriteLine("erase(begin()) = [{0} {1}]",
        it->first, it->second);

    // add elements and display " b c d e"
    c1.insert(cliext::map<wchar_t, int>::make_value(L'd', 4));
    c1.insert(cliext::map<wchar_t, int>::make_value(L'e', 5));
    for each (cliext::map<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase all but end
    it = c1.end();
    it = c1.erase(c1.begin(), --it);
    System::Console::WriteLine("erase(begin(), end()-1) = [{0} {1}]",
        it->first, it->second);
    System::Console::WriteLine("size() = {0}", c1.size());

    // erase end
    System::Console::WriteLine("erase(L'x') = {0}", c1.erase(L'x'));
    System::Console::WriteLine("erase(L'e') = {0}", c1.erase(L'e'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
erase(begin()) = [b 2]
[b 2] [c 3] [d 4] [e 5]
erase(begin(), end()-1) = [e 5]
size() = 1
erase(L'x') = 0
erase(L'e') = 1
```

## map::find (STL/CLR)

Finds an element that matches a specified key.

### Syntax

```
iterator find(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

If at least one element in the controlled sequence has equivalent ordering with *key*, the member function returns an iterator designating one of those elements; otherwise it returns [map::end \(STL/CLR\) \(\)](#). You use it to locate an element currently in the controlled sequence that matches a specified key.

### Example

```
// cliext_map_find.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());

    Mymap::iterator it = c1.find(L'b');
    System::Console::WriteLine("find {0} = [{1} {2}]",
        L'b', it->first, it->second);

    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
find A = False
find b = [b 2]
find C = False
```

## map::generic\_container (STL/CLR)

The type of the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::
    ITree<GKey, GValue>
generic_container;
```

### Remarks

The type describes the generic interface for this template container class.

### Example

```
// cliext_map_generic_container.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymap::generic_container^ gc1 = %c1;
    for each (Mymap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(Mymap::make_value(L'd', 4));
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(Mymap::make_value(L'e', 5));
    for each (Mymap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3] [d 4] [e 5]
```

## map::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

### Example

```
// cliext_map_generic_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymap::generic_container^ gc1 = %c1;
    for each (Mymap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Mymap::generic_iterator gcit = gc1->begin();
    Mymap::generic_value gcval = *gcit;
    System::Console::Write("[{0} {1}] ", gcval->first, gcval->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]
```

## map::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::  
    ReverseRandomAccessIterator<generic_value>  
    generic_reverse_iterator;
```

## Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

## Example

```
// cliext_map_generic_reverse_iterator.cpp  
// compile with: /clr  
#include <cliext/map>  
  
typedef cliext::map<wchar_t, int> Mymap;  
int main()  
{  
    Mymap c1;  
    c1.insert(Mymap::make_value(L'a', 1));  
    c1.insert(Mymap::make_value(L'b', 2));  
    c1.insert(Mymap::make_value(L'c', 3));  
  
    // display contents " [a 1] [b 2] [c 3]"  
    for each (Mymap::value_type elem in c1)  
        System::Console::Write("{0} {1}", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // construct a generic container  
    Mymap::generic_container^ gc1 = %c1;  
    for each (Mymap::value_type elem in gc1)  
        System::Console::Write("{0} {1}", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // get an element and display it  
    Mymap::generic_reverse_iterator gcit = gc1->rbegin();  
    Mymap::generic_value gcval = *gcit;  
    System::Console::WriteLine("{0} {1}", gcval->first, gcval->second);  
    return (0);  
}
```

```
[a 1] [b 2] [c 3]  
[a 1] [b 2] [c 3]  
[c 3]
```

## map::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

## Syntax

```
typedef GValue generic_value;
```

## Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

## Example

```

// cliext_map_generic_value.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymap::generic_container^ gc1 = %c1;
    for each (Mymap::value_type elem in gc1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Mymap::generic_iterator gcit = gc1->begin();
    Mymap::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} {1} ", gcval->first, gcval->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]

```

## map::insert (STL/CLR)

Adds elements.

### Syntax

```

cliext::pair<iterator, bool> insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);

```

### Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Key value to insert.

*where*

Where in container to insert (hint only).

## Remarks

Each of the member functions inserts a sequence specified by the remaining operands.

The first member function endeavors to insert an element with value *val*, and returns a pair of values `x`. If `x.second` is true, `x.first` designates the newly inserted element; otherwise `x.first` designates an element with equivalent ordering that already exists and no new element is inserted. You use it to insert a single element.

The second member function inserts an element with value *val*, using *where* as a hint (to improve performance), and returns an iterator that designates the newly inserted element. You use it to insert a single element which might be adjacent to an element you know.

The third member function inserts the sequence `[first, last]`. You use it to insert zero or more elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

Each element insertion takes time proportional to the logarithm of the number of elements in the controlled sequence. Insertion can occur in amortized constant time, however, given a hint that designates an element adjacent to the insertion point.

## Example

```

// cliext_map_insert.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
typedef Mymap::pair_iter_bool Pairib;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    // insert a single value, success and failure
    Pairib pair1 = c1.insert(Mymap::make_value(L'x', 24));
    System::Console::WriteLine("insert([L'x' 24]) = [{0} {1}] {2}",
        pair1.first->first, pair1.first->second, pair1.second);

    pair1 = c1.insert(Mymap::make_value(L'b', 2));
    System::Console::WriteLine("insert([L'b' 2]) = [{0} {1}] {2}",
        pair1.first->first, pair1.first->second, pair1.second);

    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value with hint
    Mymap::iterator it =
        c1.insert(c1.begin(), Mymap::make_value(L'y', 25));
    System::Console::WriteLine("insert(begin(), [L'y' 25]) = [{0} {1}]",
        it->first, it->second);
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an iterator range
    Mymap c2;
    it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (Mymap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an enumeration
    Mymap c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::
        IEnumerable<Mymap::value_type>)c1);
    for each (Mymap::value_type elem in c3)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
insert([L'x' 24]) = [[x 24] True]
insert([L'b' 2]) = [[b 2] False]
[a 1] [b 2] [c 3] [x 24]
insert(begin(), [L'y' 25]) = [y 25]
[a 1] [b 2] [c 3] [x 24] [y 25]
[a 1] [b 2] [c 3] [x 24]
[a 1] [b 2] [c 3] [x 24] [y 25]
```

## map::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

### Remarks

The type describes an object of unspecified type `T1` that can serve as a bidirectional iterator for the controlled sequence.

### Example

```
// cliext_map_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents "[a 1] [b 2] [c 3]"
    Mymap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("[{0} {1}] ", it->first, it->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

## map::key\_comp (STL/CLR)

Copies the ordering delegate for two keys.

### Syntax

```
key_compare^key_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two keys.

## Example

```
// cliext_map_key_comp.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    Mymap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}
```

```
compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True
```

## map::key\_compare (STL/CLR)

The ordering delegate for two keys.

### Syntax

```
Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;
```

### Remarks

The type is a synonym for the delegate that determines the ordering of its key arguments.

## Example

```

// cliext_map_key_compare.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    Mymap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## map::key\_type (STL/CLR)

The type of an ordering key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter *Key*.

### Example

```

// cliext_map_key_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using key_type
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Mymap::key_type val = it->first;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## map::lower\_bound (STL/CLR)

Finds beginning of range that matches a specified key.

### Syntax

```
iterator lower_bound(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the first element *x* in the controlled sequence that has equivalent ordering to *key*. If no such element exists, it returns [map::end \(STL/CLR\)](#) (); otherwise it returns an iterator that designates *x*. You use it to locate the beginning of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_map_lower_bound.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    Mymap::iterator it = c1.lower_bound(L'a');
    System::Console::WriteLine("*lower_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.lower_bound(L'b');
    System::Console::WriteLine("*lower_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
lower_bound(L'x') == end() = True
*lower_bound(L'a') = [a 1]
*lower_bound(L'b') = [b 2]

```

## map::make\_value (STL/CLR)

Constructs a value object.

### Syntax

```
static value_type make_value(key_type key, mapped_type mapped);
```

### Parameters

*key*

Key value to use.

*mapped*

Mapped value to search for.

### Remarks

The member function returns a `value_type` object whose key is *key* and whose mapped value is *mapped*. You use it to compose an object suitable for use with several other member functions.

### Example

```

// cliext_map_make_value.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## map::map (STL/CLR)

Constructs a container object.

### Syntax

```

map();
explicit map(key_compare^ pred);
map(map<Key, Mapped>% right);
map(map<Key, Mapped>^ right);
template<typename InIter>
    mapmap(InIter first, InIter last);
template<typename InIter>
    map(InIter first, InIter last,
        key_compare^ pred);
map(System::Collections::Generic::IEnumerable<GValue>^ right);
map(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);

```

### Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

### Remarks

The constructor:

`map();`

initializes the controlled sequence with no elements, with the default ordering predicate `key_compare()`. You use

it to specify an empty initial controlled sequence, with the default ordering predicate.

The constructor:

```
explicit map(key_compare^ pred);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate.

The constructor:

```
map(map<Key, Mapped>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`], with the default ordering predicate. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the map object *right*, with the default ordering predicate.

The constructor:

```
map(map<Key, Mapped>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`], with the default ordering predicate. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the map object *right*, with the default ordering predicate.

The constructor:

```
template<typename InIter> map(InIter first, InIter last);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the default ordering predicate. You use it to make the controlled sequence a copy of another sequence, with the default ordering predicate.

The constructor:

```
template<typename InIter> map(InIter first, InIter last, key_compare^ pred);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate.

The constructor:

```
map(System::Collections::Generic::IEnumerable<Key>% right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the default ordering predicate. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the default ordering predicate.

The constructor:

```
map(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate.

## Example

```
// cliext_map_construct.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
```

```

{
// construct an empty container
Mymap c1;
System::Console::WriteLine("size() = {0}", c1.size());

c1.insert(Mymap::make_value(L'a', 1));
c1.insert(Mymap::make_value(L'b', 2));
c1.insert(Mymap::make_value(L'c', 3));
for each (Mymap::value_type elem in c1)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an ordering rule
Mymap c2 = cliext::greater_equal<wchar_t>();
System::Console::WriteLine("size() = {0}", c2.size());

c2.insert(c1.begin(), c1.end());
for each (Mymap::value_type elem in c2)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range
Mymap c3(c1.begin(), c1.end());
for each (Mymap::value_type elem in c3)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Mymap c4(c1.begin(), c1.end(),
          cliext::greater_equal<wchar_t>());
for each (Mymap::value_type elem in c4)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration
Mymap c5( // NOTE: cast is not needed
          (System::Collections::Generic::IEnumerable<
            Mymap::value_type>^)%c3);
for each (Mymap::value_type elem in c5)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Mymap c6( // NOTE: cast is not needed
          (System::Collections::Generic::IEnumerable<
            Mymap::value_type>^)%c3,
          cliext::greater_equal<wchar_t>());
for each (Mymap::value_type elem in c6)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying another container
Mymap c7(c4);
for each (Mymap::value_type elem in c7)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying a container handle
Mymap c8(%c3);
for each (Mymap::value_type elem in c8)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
```

## map::mapped\_type (STL/CLR)

The type of a mapped value associated with each key.

### Syntax

```
typedef Mapped mapped_type;
```

### Remarks

The type is a synonym for the template parameter *Mapped*.

### Example

```
// cliext_map_mapped_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using mapped_type
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in mapped_type object
        Mymap::mapped_type val = it->second;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
1 2 3
```

## map::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
map<Key, Mapped>% operator=(map<Key, Mapped>% right);
```

## Parameters

*right*

Container to copy.

## Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

## Example

```
// cliext_map_operator_as.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2 = c1;
    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

## map::operator(STL/CLR)

Maps a key to its associated mapped value.

## Syntax

```
mapped_type operator[](key_type key);
```

## Parameters

*key*

Key value to search for.

## Remarks

The member functions endeavors to find an element with equivalent ordering to *key*. If it finds one, it returns the associated mapped value; otherwise, it inserts `value_type(key, mapped_type())` and returns the associated (default) mapped value. You use it to look up a mapped value given its associated key, or to ensure that an entry exists for the key if none is found.

## Example

```

// cliext_map_operator_sub.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("c1[{0}] = {1}",
        L'A', c1[L'A']);
    System::Console::WriteLine("c1[{0}] = {1}",
        L'b', c1[L'b']);

    // redisplay altered contents
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // alter mapped values and redisplay
    c1[L'A'] = 10;
    c1[L'c'] = 13;
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
c1[A] = 0
c1[b] = 2
[A 0] [a 1] [b 2] [c 3]
[A 10] [a 1] [b 2] [c 13]

```

## map::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

### Syntax

```
reverse_iterator rbegin();
```

### Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the **beginning** of the reverse sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_map_rbegin.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymap::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("++rbegin() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*rbegin() = [c 3]
*++rbegin() = [b 2]

```

## map::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_map_reference.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
    {   // get a reference to an element
        Mymap::reference ref = *it;
        System::Console::Write("[{0} {1}] ", ref->first, ref->second);
    }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## map::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

### Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_map_rend.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymap::reverse_iterator rit = c1.rend();
    --rit;
    --rit;
    System::Console::WriteLine("---- --rend() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("----rend() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
---- --rend() = [b 2]
----rend() = [a 1]

```

## map::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

### Syntax

```
typedef T3 reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

### Example

```
// cliext_map_reverse_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Mymap::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("[{0} {1}] ", rit->first, rit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

## map::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [map::empty \(STL/CLR\) \(\)](#).

### Example

```

// cliext_map_size.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(Mymap::make_value(L'd', 4));
    c1.insert(Mymap::make_value(L'e', 5));
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0 after clearing
size() = 2 after adding 2

```

## map::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_map_size_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Mymap::size_type diff = 0;
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3

```

## map::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```

void swap(map<Key, Mapped>% right);

```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_map_swap.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Mymap c2;
    c2.insert(Mymap::make_value(L'd', 4));
    c2.insert(Mymap::make_value(L'e', 5));
    c2.insert(Mymap::make_value(L'f', 6));
    for each (Mymap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    for each (Mymap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[d 4] [e 5] [f 6]
[d 4] [e 5] [f 6]
[a 1] [b 2] [c 3]

```

## map::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<value_type>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_map_to_array.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // copy the container and modify it
    cli::array<Mymap::value_type>^ a1 = c1.to_array();

    c1.insert(Mymap::make_value(L'd', 4));
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (Mymap::value_type elem in a1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3]

```

## map::upper\_bound (STL/CLR)

Finds end of range that matches a specified key.

### Syntax

```
iterator upper_bound(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the last element *x* in the controlled sequence that has equivalent ordering to *key*. If no such element exists, or if *x* is the last element in the controlled sequence, it returns `map::end` (STL/CLR)(); otherwise it returns an iterator that designates the first element beyond *x*. You use it to locate the end of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_map_upper_bound.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    Mymap::iterator it = c1.upper_bound(L'a');
    System::Console::WriteLine("*upper_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.upper_bound(L'b');
    System::Console::WriteLine("*upper_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
upper_bound(L'x') == end() = True
*upper_bound(L'a') = [b 2]
*upper_bound(L'b') = [c 3]

```

## map::value\_comp (STL/CLR)

Copies the ordering delegate for two element values.

### Syntax

```

value_compare^ value_comp();

```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two element values.

### Example

```

// cliext_map_value_comp.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    Mymap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Mymap::make_value(L'a', 1),
            Mymap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Mymap::make_value(L'a', 1),
            Mymap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Mymap::make_value(L'b', 2),
            Mymap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}

```

```

compare([L'a', 1], [L'a', 1]) = False
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False

```

## map::value\_compare (STL/CLR)

The ordering delegate for two element values.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
value_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its value arguments.

### Example

```

// cliext_map_value_compare.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    Mymap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Mymap::make_value(L'a', 1),
            Mymap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Mymap::make_value(L'a', 1),
            Mymap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Mymap::make_value(L'b', 2),
            Mymap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}

```

```

compare([L'a', 1], [L'a', 1]) = False
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False

```

## map::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef generic_value value_type;
```

### Remarks

The type is a synonym for `generic_value`.

### Example

```

// cliext_map_value_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using value_type
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Mymap::value_type val = *it;
        System::Console::Write("[{0} {1}] ", val->first, val->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## operator!= (map) (STL/CLR)

List not equal comparison.

### Syntax

```

template<typename Key,
         typename Mapped>
bool operator!=(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two maps are compared element by element.

### Example

```

// cliext_map_operator_ne.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

## operator< (map) (STL/CLR)

List less than comparison.

### Syntax

```

template<typename Key,
         typename Mapped>
bool operator<(map<Key, Mapped>% left,
                 map<Key, Mapped>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true if, for the lowest position *i* for which  $\text{right}[i] < \text{left}[i]$  it is also true

that `left[i] < right[i]`. Otherwise, it returns `left->size() < right->size()`. You use it to test whether *left* is ordered before *right* when the two maps are compared element by element.

## Example

```
// cliext_map_operator_lt.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

## operator<= (map) (STL/CLR)

List less than or equal comparison.

### Syntax

```
template<typename Key,
         typename Mapped>
bool operator<=(map<Key, Mapped>% left,
                 map<Key, Mapped>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two maps are compared element by element.

## Example

```
// cliext_map_operator_le.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator== (map) (STL/CLR)

List equal comparison.

## Syntax

```
template<typename Key,
         typename Mapped>
bool operator==(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position *i*, *left[i] == right[i]*. You use it to test whether *left* is ordered the same as *right* when the two maps are compared element by element.

## Example

```
// cliext_map_operator_eq.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents "[a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents "[a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

## operator> (map) (STL/CLR)

List greater than comparison.

## Syntax

```
template<typename Key,
         typename Mapped>
bool operator>(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two maps are compared element by element.

## Example

```
// cliext_map_operator_gt.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
                             c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
                             c2 > c1);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

`operator>= (map) (STL/CLR)`

List greater than or equal comparison.

## Syntax

```
template<typename Key,
         typename Mapped>
bool operator>=(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two maps are compared element by element.

### Example

```
// cliext_map_operator_ge.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

# multimap (STL/CLR)

9/20/2022 • 42 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `multimap` to manage a sequence of elements as a (nearly) balanced ordered tree of nodes, each storing one element. An element consists of a key, for ordering the sequence, and a mapped value, which goes along for the ride.

In the description below, `GValue` is the same as:

`Microsoft::VisualC::StlClr::GenericPair<GKey, GMapped>`

where:

`GKey` is the same as *Key* unless the latter is a ref type, in which case it is `Key^`

`GMapped` is the same as *Mapped* unless the latter is a ref type, in which case it is `Mapped^`

## Syntax

```
template<typename Key,
         typename Mapped>
ref class multimap
{
    : public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    Microsoft::VisualC::StlClr::ITree<Gkey, GValue>
{ .... };
```

## Parameters

### *Key*

The type of the key component of an element in the controlled sequence.

### *Mapped*

The type of the additional component of an element in the controlled sequence.

## Requirements

**Header:** `<cliext/map>`

**Namespace:** `cliext`

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">multimap::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.
<a href="#">multimap::const_reference (STL/CLR)</a>	The type of a constant reference to an element.

TYPE DEFINITION	DESCRIPTION
<code>multimap::const_reverse_iterator (STL/CLR)</code>	The type of a constant reverse iterator for the controlled sequence.
<code>multimap::difference_type (STL/CLR)</code>	The type of a (possibly signed) distance between two elements.
<code>multimap::generic_container (STL/CLR)</code>	The type of the generic interface for the container.
<code>multimap::generic_iterator (STL/CLR)</code>	The type of an iterator for the generic interface for the container.
<code>multimap::generic_reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the generic interface for the container.
<code>multimap::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container.
<code>multimap::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>multimap::key_compare (STL/CLR)</code>	The ordering delegate for two keys.
<code>multimap::key_type (STL/CLR)</code>	The type of an ordering key.
<code>multimap::mapped_type (STL/CLR)</code>	The type of the mapped value associated with each key.
<code>multimap::reference (STL/CLR)</code>	The type of a reference to an element.
<code>multimap::reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the controlled sequence.
<code>multimap::size_type (STL/CLR)</code>	The type of a (non-negative) distance between two elements.
<code>multimap::value_compare (STL/CLR)</code>	The ordering delegate for two element values.
<code>multimap::value_type (STL/CLR)</code>	The type of an element.

MEMBER FUNCTION	DESCRIPTION
<code>multimap::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>multimap::clear (STL/CLR)</code>	Removes all elements.
<code>multimap::count (STL/CLR)</code>	Counts elements matching a specified key.
<code>multimap::empty (STL/CLR)</code>	Tests whether no elements are present.
<code>multimap::end (STL/CLR)</code>	Designates the end of the controlled sequence.
<code>multimap::equal_range (STL/CLR)</code>	Finds range that matches a specified key.
<code>multimap::erase (STL/CLR)</code>	Removes elements at specified positions.

MEMBER FUNCTION	DESCRIPTION
<code>multimap::find (STL/CLR)</code>	Finds an element that matches a specified key.
<code>multimap::insert (STL/CLR)</code>	Adds elements.
<code>multimap::key_comp (STL/CLR)</code>	Copies the ordering delegate for two keys.
<code>multimap::lower_bound (STL/CLR)</code>	Finds beginning of range that matches a specified key.
<code>multimap::make_value (STL/CLR)</code>	Constructs a value object.
<code>multimap::multimap (STL/CLR)</code>	Constructs a container object.
<code>multimap::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.
<code>multimap::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>multimap::size (STL/CLR)</code>	Counts the number of elements.
<code>multimap::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>multimap::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>multimap::upper_bound (STL/CLR)</code>	Finds end of range that matches a specified key.
<code>multimap::value_comp (STL/CLR)</code>	Copies the ordering delegate for two element values.
OPERATOR	DESCRIPTION
<code>multimap::operator= (STL/CLR)</code>	Replaces the controlled sequence.
<code>operator!= (multimap) (STL/CLR)</code>	Determines if a <code>multimap</code> object is not equal to another <code>multimap</code> object.
<code>operator&lt; (multimap) (STL/CLR)</code>	Determines if a <code>multimap</code> object is less than another <code>multimap</code> object.
<code>operator&lt;= (multimap) (STL/CLR)</code>	Determines if a <code>multimap</code> object is less than or equal to another <code>multimap</code> object.
<code>operator== (multimap) (STL/CLR)</code>	Determines if a <code>multimap</code> object is equal to another <code>multimap</code> object.
<code>operator&gt; (multimap) (STL/CLR)</code>	Determines if a <code>multimap</code> object is greater than another <code>multimap</code> object.
<code>operator&gt;= (multimap) (STL/CLR)</code>	Determines if a <code>multimap</code> object is greater than or equal to another <code>multimap</code> object.

## Interfaces

INTERFACE	DESCRIPTION
<a href="#">ICloneable</a>	Duplicate an object.
<a href="#">IEnumerable</a>	Sequence through elements.
<a href="#">ICollection</a>	Maintain group of elements.
<a href="#">IEnumerable&lt;T&gt;</a>	Sequence through typed elements.
<a href="#">ICollection&lt;T&gt;</a>	Maintain group of typed elements.
<a href="#">ITree&lt;Key, Value&gt;</a>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes. It inserts elements into a (nearly) balanced tree that it keeps ordered by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements.

The object orders the sequence it controls by calling a stored delegate object of type [multimap::key\\_compare \(STL/CLR\)](#). You can specify the stored delegate object when you construct the multimap; if you specify no delegate object, the default is the comparison `operator<(key_type, key_type)`. You access this stored object by calling the member function [multimap::key\\_comp \(STL/CLR\)](#) () .

Such a delegate object must impose a strict weak ordering on keys of type [multimap::key\\_type \(STL/CLR\)](#). That means, for any two keys `x` and `y`:

`key_comp()(x, y)` returns the same Boolean result on every call.

If `key_comp()(x, y)` is true, then `key_comp()(y, x)` must be false.

If `key_comp()(x, y)` is true, then `x` is said to be ordered before `y`.

If `!key_comp()(x, y) && !key_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

For any element `x` that precedes `y` in the controlled sequence, `key_comp()(y, x)` is false. (For the default delegate object, keys never decrease in value.) Unlike template class [map \(STL/CLR\)](#), an object of template class [multimap](#) does not require that keys for all elements are unique. (Two or more keys can have equivalent ordering.)

Each element contains a separate key and a mapped value. The sequence is represented in a way that permits lookup, insertion, and removal of an arbitrary element with a number of operations proportional to the logarithm of the number of elements in the sequence (logarithmic time). Moreover, inserting an element invalidates no iterators, and removing an element invalidates only those iterators which point at the removed element.

A multimap supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by [multimap::end \(STL/CLR\)](#) (). You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a multimap iterator to reach the head node, and it will then compare equal to `end()`. But you cannot dereference the iterator returned by `end()`.

Note that you cannot refer to a multimap element directly given its numerical position -- that requires a random-access iterator.

A multimap iterator stores a handle to its associated multimap node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A multimap iterator remains valid so long as its associated multimap node is associated with some multimap. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### `multimap::begin (STL/CLR)`

Designates the beginning of the controlled sequence.

#### Syntax

```
iterator begin();
```

#### Remarks

The member function returns a bidirectional iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

#### Example

```
// cliext_multimap_begin.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items
    Mymultimap::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("**begin() = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*begin() = [a 1]
*++begin() = [b 2]
```

## multimap::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

### Remarks

The member function effectively calls [multimap::erase \(STL/CLR\)](#)([multimap::begin \(STL/CLR\)](#)([\(\)](#), [multimap::end \(STL/CLR\)](#)([\(\)](#))). You use it to ensure that the controlled sequence is empty.

### Example

```
// cliext_multimap_clear.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents "[a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));

    // display contents "[a 1] [b 2]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2]
size() = 0
```

## multimap::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

## Syntax

```
typedef T2 const_iterator;
```

## Remarks

The type describes an object of unspecified type `T2` that can serve as a constant bidirectional iterator for the controlled sequence.

## Example

```
// cliext_multimap_const_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymultimap::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        System::Console::Write("[{0} {1}] ", cit->first, cit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

## multimap::const\_reference (STL/CLR)

The type of a constant reference to an element.

## Syntax

```
typedef value_type% const_reference;
```

## Remarks

The type describes a constant reference to an element.

## Example

```

// cliext_multimap_const_reference.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymultimap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Mymultimap::const_reference cref = *cit;
        System::Console::Write("[{0} {1}] ", cref->first, cref->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## multimap::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence.

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```

// cliext_multimap_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Mymultimap::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("[{0} {1}] ", crit->first, crit->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[c 3] [b 2] [a 1]
```

## multimap::count (STL/CLR)

Finds the number of elements matching a specified key.

### Syntax

```
size_type count(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns the number of elements in the controlled sequence that have equivalent ordering with *key*. You use it to determine the number of elements currently in the controlled sequence that match a specified key.

### Example

```
// cliext_multimap_count.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

## multimap::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

## Remarks

The type describes a possibly negative element count.

## Example

```
// cliext_multimap_difference_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents "[a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Mymultimap::difference_type diff = 0;
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Mymultimap::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
end()-begin() = 3
begin()-end() = -3
```

## multimap::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

## Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [multimap::size \(STL/CLR\) \(\) == 0](#). You use it to test whether the multimap is empty.

## Example

```

// cliext_multimap_empty.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 3
empty() = False
size() = 0
empty() = True

```

## multimap::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a bidirectional iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_multimap_end.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect last two items
    Mymultimap::iterator it = c1.end();
    --it;
    --it;
    System::Console::WriteLine("---- --end() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("----end() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
---- --end() = [b 2]
----end() = [c 3]

```

## multimap::equal\_range (STL/CLR)

Finds range that matches a specified key.

### Syntax

```

pair_iter_iter equal_range(key_type _Keyval);

```

### Parameters

*\_Keyval*

Key value to search for.

### Remarks

The method returns a pair of iterators `- multimap::lower_bound (STL/CLR) (_Keyval), multimap::upper_bound (STL/CLR) (_Keyval)`. You use it to determine the range of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_multimap_equal_range.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
typedef Mymultimap::pair_iter_iter Pairii;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("[{0} {1}] ",
            pair1.first->first, pair1.first->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
equal_range(L'x') empty = True
[b 2]

```

## multimap::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

### Parameters

*first*

Beginning of range to erase.

*key*

Key value to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*, and returns an iterator that designates the first element remaining beyond the element removed, or [multimap::end \(STL/CLR\)](#)  
() if no such element exists. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [ *first* , *last* ), and returns an iterator that designates the first element remaining beyond any elements removed, or *end()* if no such element exists.. You use it to remove zero or more contiguous elements.

The third member function removes any element of the controlled sequence whose key has equivalent ordering to *key*, and returns a count of the number of elements removed. You use it to remove and count all elements that match a specified key.

Each element erasure takes time proportional to the logarithm of the number of elements in the controlled sequence.

## Example

```
// cliext_multimap_erase.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    cliext::multimap<wchar_t, int> c1;
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'a', 1));
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'b', 2));
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (cliext::multimap<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase an element and reinspect
    cliext::multimap<wchar_t, int>::iterator it =
        c1.erase(c1.begin());
    System::Console::WriteLine("erase(begin()) = [{0} {1}]",
        it->first, it->second);

    // add elements and display " b c d e"
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'd', 4));
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'e', 5));
    for each (cliext::multimap<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase all but end
    it = c1.end();
    it = c1.erase(c1.begin(), --it);
    System::Console::WriteLine("erase(begin(), end()-1) = [{0} {1}]",
        it->first, it->second);
    System::Console::WriteLine("size() = {0}", c1.size());

    // erase end
    System::Console::WriteLine("erase(L'x') = {0}", c1.erase(L'x'));
    System::Console::WriteLine("erase(L'e') = {0}", c1.erase(L'e'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
erase(begin()) = [b 2]
[b 2] [c 3] [d 4] [e 5]
erase(begin(), end()-1) = [e 5]
size() = 1
erase(L'x') = 0
erase(L'e') = 1
```

## multimap::find (STL/CLR)

Finds an element that matches a specified key.

### Syntax

```
iterator find(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

If at least one element in the controlled sequence has equivalent ordering with *key*, the member function returns an iterator designating one of those elements; otherwise it returns [multimap::end \(STL/CLR\)](#) (). You use it to locate an element currently in the controlled sequence that matches a specified key.

### Example

```
// cliext_multimap_find.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());

    Mymultimap::iterator it = c1.find(L'b');
    System::Console::WriteLine("find {0} = [{1} {2}]",
        L'b', it->first, it->second);

    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
find A = False
find b = [b 2]
find C = False
```

## multimap::generic\_container (STL/CLR)

The type of the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::
    ITree<GKey, GValue>
generic_container;
```

### Remarks

The type describes the generic interface for this template container class.

### Example

```
// cliext_multimap_generic_container.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymultimap::generic_container^ gc1 = %c1;
    for each (Mymultimap::value_type elem in gc1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(Mymultimap::make_value(L'd', 4));
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(Mymultimap::make_value(L'e', 5));
    for each (Mymultimap::value_type elem in gc1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3] [d 4] [e 5]
```

## multimap::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

### Example

```
// cliext_multimap_generic_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymultimap::generic_container^ gc1 = %c1;
    for each (Mymultimap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Mymultimap::generic_iterator gcit = gc1->begin();
    Mymultimap::generic_value gcval = *gcit;
    System::Console::Write("[{0} {1}] ", gcval->first, gcval->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]
```

## multimap::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::  
    ReverseRandomAccessIterator<generic_value>  
    generic_reverse_iterator;
```

## Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

## Example

```
// cliext_multimap_generic_reverse_iterator.cpp  
// compile with: /clr  
#include <cliext/map>  
  
typedef cliext::multimap<wchar_t, int> Mymultimap;  
int main()  
{  
    Mymultimap c1;  
    c1.insert(Mymultimap::make_value(L'a', 1));  
    c1.insert(Mymultimap::make_value(L'b', 2));  
    c1.insert(Mymultimap::make_value(L'c', 3));  
  
    // display contents " [a 1] [b 2] [c 3]"  
    for each (Mymultimap::value_type elem in c1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // construct a generic container  
    Mymultimap::generic_container^ gc1 = %c1;  
    for each (Mymultimap::value_type elem in gc1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // get an element and display it  
    Mymultimap::generic_reverse_iterator gcit = gc1->rbegin();  
    Mymultimap::generic_value gcval = *gcit;  
    System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);  
    return (0);  
}
```

```
[a 1] [b 2] [c 3]  
[a 1] [b 2] [c 3]  
[c 3]
```

## multimap::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

## Syntax

```
typedef GValue generic_value;
```

## Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

## Example

```

// cliext_multimap_generic_value.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymultimap::generic_container^ gc1 = %c1;
    for each (Mymultimap::value_type elem in gc1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Mymultimap::generic_iterator gcit = gc1->begin();
    Mymultimap::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} {1} ", gcval->first, gcval->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]

```

## multimap::insert (STL/CLR)

Adds elements.

### Syntax

```

iterator insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);

```

### Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Key value to insert.

*where*

Where in container to insert (hint only).

### Remarks

Each of the member functions inserts a sequence specified by the remaining operands.

The first member function inserts an element with value *val*, and returns an iterator that designates the newly inserted element. You use it to insert a single element.

The second member function inserts an element with value *val*, using *where* as a hint (to improve performance), and returns an iterator that designates the newly inserted element. You use it to insert a single element which might be adjacent to an element you know.

The third member function inserts the sequence [`first`, `last`]. You use it to insert zero or more elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

Each element insertion takes time proportional to the logarithm of the number of elements in the controlled sequence. Insertion can occur in amortized constant time, however, given a hint that designates an element adjacent to the insertion point.

### Example

```

// cliext_multimap_insert.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Mymultimap::iterator it =
        c1.insert(Mymultimap::make_value(L'x', 24));
    System::Console::WriteLine("insert([L'x' 24]) = [{0} {1}]",
        it->first, it->second);

    it = c1.insert(Mymultimap::make_value(L'b', 2));
    System::Console::WriteLine("insert([L'b' 2]) = [{0} {1}]",
        it->first, it->second);

    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value with hint
    it = c1.insert(c1.begin(), Mymultimap::make_value(L'y', 25));
    System::Console::WriteLine("insert(begin(), [L'y' 25]) = [{0} {1}]",
        it->first, it->second);
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an iterator range
    Mymultimap c2;
    it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an enumeration
    Mymultimap c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::
        IEnumerable<Mymultimap::value_type>)c1);
    for each (Mymultimap::value_type elem in c3)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
insert([L'x' 24]) = [x 24]
insert([L'b' 2]) = [b 2]
[a 1] [b 2] [b 2] [c 3] [x 24]
insert(begin(), [L'y' 25]) = [y 25]
[a 1] [b 2] [b 2] [c 3] [x 24] [y 25]
[a 1] [b 2] [b 2] [c 3] [x 24]
[a 1] [b 2] [b 2] [c 3] [x 24] [y 25]
```

## multimap::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

### Remarks

The type describes an object of unspecified type `T1` that can serve as a bidirectional iterator for the controlled sequence.

### Example

```
// cliext_multimap_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymultimap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("[{0} {1}] ", it->first, it->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

## multimap::key\_comp (STL/CLR)

Copies the ordering delegate for two keys.

### Syntax

```
key_compare^key_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two keys.

## Example

```
// cliext_multimap_key_comp.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    Mymultimap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymultimap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}
```

```
compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True
```

## multimap::key\_compare (STL/CLR)

The ordering delegate for two keys.

### Syntax

```
Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;
```

### Remarks

The type is a synonym for the delegate that determines the ordering of its key arguments.

## Example

```

// cliext_multimap_key_compare.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    Mymultimap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymultimap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## multimap::key\_type (STL/CLR)

The type of an ordering key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter *Key*.

### Example

```

// cliext_multimap_key_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using key_type
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Mymultimap::key_type val = it->first;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## multimap::lower\_bound (STL/CLR)

Finds beginning of range that matches a specified key.

### Syntax

```
iterator lower_bound(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the first element  $x$  in the controlled sequence that has equivalent ordering to *key*. If no such element exists, it returns [multimap::end \(STL/CLR\)](#) (); otherwise it returns an iterator that designates  $x$ . You use it to locate the beginning of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_multimap_lower_bound.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    Mymultimap::iterator it = c1.lower_bound(L'a');
    System::Console::WriteLine("*lower_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.lower_bound(L'b');
    System::Console::WriteLine("*lower_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
lower_bound(L'x') == end() = True
*lower_bound(L'a') = [a 1]
*lower_bound(L'b') = [b 2]

```

## multimap::make\_value (STL/CLR)

Constructs a value object.

### Syntax

```
static value_type make_value(key_type key, mapped_type mapped);
```

### Parameters

*key*

Key value to use.

*mapped*

Mapped value to search for.

### Remarks

The member function returns a `value_type` object whose key is *key* and whose mapped value is *mapped*. You use it to compose an object suitable for use with several other member functions.

### Example

```

// cliext_multimap_make_value.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## multimap::mapped\_type (STL/CLR)

The type of a mapped value associated with each key.

### Syntax

```
typedef Mapped mapped_type;
```

### Remarks

The type is a synonym for the template parameter *Mapped*.

### Example

```

// cliext_multimap_mapped_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using mapped_type
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in mapped_type object
        Mymultimap::mapped_type val = it->second;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

1 2 3

# multimap::multimap (STL/CLR)

Constructs a container object.

## Syntax

```
multimap();
explicit multimap(key_compare^ pred);
multimap(multimap<Key, Mapped>% right);
multimap(multimap<Key, Mapped>^ right);
template<typename InIter>
    multimap(multimap<InIter first, InIter last>);
template<typename InIter>
    multimap(InIter first, InIter last,
        key_compare^ pred);
multimap(System::Collections::Generic::IEnumerable<GValue>^ right);
multimap(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);
```

## Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

## Remarks

The constructor:

```
multimap();
```

initializes the controlled sequence with no elements, with the default ordering predicate `key_compare()`. You use it to specify an empty initial controlled sequence, with the default ordering predicate.

The constructor:

```
explicit multimap(key_compare^ pred);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate.

The constructor:

```
multimap(multimap<Key, Mapped>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`], with the default ordering predicate. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the `multimap` object *right*, with the default ordering predicate.

The constructor:

```
multimap(multimap<Key, Mapped>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`], with the default ordering predicate. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the

multimap object *right*, with the default ordering predicate.

The constructor:

```
template<typename InIter> multimap(InIter first, InIter last);
```

initializes the controlled sequence with the sequence [*first*, *last*), with the default ordering predicate. You use it to make the controlled sequence a copy of another sequence, with the default ordering predicate.

The constructor:

```
template<typename InIter> multimap(InIter first, InIter last, key_compare^ pred);
```

initializes the controlled sequence with the sequence [*first*, *last*), with the ordering predicate *pred*. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate.

The constructor:

```
multimap(System::Collections::Generic::IEnumerable<Key>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the default ordering predicate. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the default ordering predicate.

The constructor:

```
multimap(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate.

## Example

```
// cliext_multimap_construct.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    //
    // construct an empty container
    Mymultimap c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    //
    // construct with an ordering rule
    Mymultimap c2 = cliext::greater_equal<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    c2.insert(c1.begin(), c1.end());
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    //
    // construct with an iterator range
    Mymultimap c3(c1.begin(), c1.end());
    for each (Mymultimap::value_type elem in c3)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
```

```

System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Mymultimap c4(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>());
for each (Mymultimap::value_type elem in c4)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration
Mymultimap c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Mymultimap::value_type>^)%c3);
for each (Mymultimap::value_type elem in c5)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Mymultimap c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Mymultimap::value_type>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (Mymultimap::value_type elem in c6)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying another container
Mymultimap c7(c4);
for each (Mymultimap::value_type elem in c7)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying a container handle
Mymultimap c8(%c3);
for each (Mymultimap::value_type elem in c8)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
return (0);
}

```

```

size() = 0
[a 1] [b 2] [c 3]
size() = 0
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]

```

## multimap::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```

multimap<Key, Mapped>% operator=(multimap<Key, Mapped>% right);

```

### Parameters

*right*

Container to copy.

## Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

## Example

```
// cliext_multimap_operator_as.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2 = c1;
    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

## multimap::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

## Syntax

```
reverse_iterator rbegin();
```

## Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the `beginning` of the reverse sequence. You use it to obtain an iterator that designates the `current` beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

## Example

```

// cliext_multimap_rbegin.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymultimap::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("++rbegin() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*rbegin() = [c 3]
*++rbegin() = [b 2]

```

## multimap::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_multimap_reference.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymultimap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        Mymultimap::reference ref = *it;
        System::Console::Write("[{0} {1}] ", ref->first, ref->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## multimap::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

### Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_multimap_rend.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymultimap::reverse_iterator rit = c1.rend();
    --rit;
    --rit;
    System::Console::WriteLine("---- --rend() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("----rend() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*-- --rend() = [b 2]
---rend() = [a 1]

```

## `multimap::reverse_iterator` (STL/CLR)

The type of a reverse iterator for the controlled sequence.

### Syntax

```
typedef T3 reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

### Example

```

// cliext_multimap_reverse_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Mymultimap::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("[{0} {1}] ", rit->first, rit->second);
    System::Console::WriteLine();
    return (0);
}

```

[c 3] [b 2] [a 1]

## **mymultimap::size (STL/CLR)**

Counts the number of elements.

### **Syntax**

```
size_type size();
```

### **Remarks**

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [mymultimap::empty \(STL/CLR\) \(\)](#).

### **Example**

```

// cliext_multimap_size.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(Mymultimap::make_value(L'd', 4));
    c1.insert(Mymultimap::make_value(L'e', 5));
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0 after clearing
size() = 2 after adding 2

```

## multimap::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_multimap_size_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Mymultimap::size_type diff = 0;
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3

```

## multimap::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```

void swap(multimap<Key, Mapped>% right);

```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_multimap_swap.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'd', 4));
    c2.insert(Mymultimap::make_value(L'e', 5));
    c2.insert(Mymultimap::make_value(L'f', 6));
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[d 4] [e 5] [f 6]
[d 4] [e 5] [f 6]
[a 1] [b 2] [c 3]

```

## multimap::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<value_type>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_multimap_to_array.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // copy the container and modify it
    cli::array<Mymultimap::value_type>^ a1 = c1.to_array();

    c1.insert(Mymultimap::make_value(L'd', 4));
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (Mymultimap::value_type elem in a1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3]

```

## multimap::upper\_bound (STL/CLR)

Finds end of range that matches a specified key.

### Syntax

```

iterator upper_bound(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the last element *x* in the controlled sequence that has equivalent ordering to *key*. If no such element exists, or if *x* is the last element in the controlled sequence, it returns [multimap::end \(STL/CLR\)](#); otherwise it returns an iterator that designates the first element beyond *x*. You use it to locate the end of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_multimap_upper_bound.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    Mymultimap::iterator it = c1.upper_bound(L'a');
    System::Console::WriteLine("*upper_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.upper_bound(L'b');
    System::Console::WriteLine("*upper_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
upper_bound(L'x') == end() = True
*upper_bound(L'a') = [b 2]
*upper_bound(L'b') = [c 3]

```

## multimap::value\_comp (STL/CLR)

Copies the ordering delegate for two element values.

### Syntax

```
value_compare^ value_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two element values.

### Example

```

// cliext_multimap_value_comp.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    Mymultimap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Mymultimap::make_value(L'a', 1),
            Mymultimap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Mymultimap::make_value(L'a', 1),
            Mymultimap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Mymultimap::make_value(L'b', 2),
            Mymultimap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}

```

```

compare([L'a', 1], [L'a', 1]) = False
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False

```

## multimap::value\_compare (STL/CLR)

The ordering delegate for two element values.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
value_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its value arguments.

### Example

```

// cliext_multimap_value_compare.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    Mymultimap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Mymultimap::make_value(L'a', 1),
            Mymultimap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Mymultimap::make_value(L'a', 1),
            Mymultimap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Mymultimap::make_value(L'b', 2),
            Mymultimap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}

```

```

compare([L'a', 1], [L'a', 1]) = False
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False

```

## multimap::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef generic_value value_type;
```

### Remarks

The type is a synonym for `generic_value`.

### Example

```

// cliext_multimap_value_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using value_type
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Mymultimap::value_type val = *it;
        System::Console::Write("[{0} {1}] ", val->first, val->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

## operator!= (multimap) (STL/CLR)

List not equal comparison.

### Syntax

```

template<typename Key,
         typename Mapped>
bool operator!=(multimap<Key, Mapped>% left,
                  multimap<Key, Mapped>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two multimaps are compared element by element.

### Example

```

// cliext_multimap_operator_ne.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

## operator< (multimap) (STL/CLR)

List less than comparison.

### Syntax

```

template<typename Key,
         typename Mapped>
bool operator<(multimap<Key, Mapped>% left,
                 multimap<Key, Mapped>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true if, for the lowest position *i* for which !(right[i] < left[i]) it is also true

that `left[i] < right[i]`. Otherwise, it returns `left->size() < right->size()`. You use it to test whether *left* is ordered before *right* when the two multimaps are compared element by element.

## Example

```
// cliext_multimap_operator_lt.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

## operator<= (multimap) (STL/CLR)

List less than or equal comparison.

### Syntax

```
template<typename Key,
         typename Mapped>
bool operator<=(multimap<Key, Mapped>% left,
                 multimap<Key, Mapped>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two multimaps are compared element by element.

## Example

```
// cliext_multimap_operator_le.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator== (multimap) (STL/CLR)

List equal comparison.

## Syntax

```
template<typename Key,
         typename Mapped>
bool operator==(multimap<Key, Mapped>% left,
                  multimap<Key, Mapped>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position *i*, *left[i] == right[i]*. You use it to test whether *left* is ordered the same as *right* when the two multimaps are compared element by element.

## Example

```
// cliext_multimap_operator_eq.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

## operator> (multimap) (STL/CLR)

List greater than comparison.

## Syntax

```
template<typename Key,
         typename Mapped>
bool operator>(multimap<Key, Mapped>% left,
                  multimap<Key, Mapped>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two multimaps are compared element by element.

### Example

```
// cliext_multimap_operator_gt.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
                             c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
                             c2 > c1);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

`operator>=` (multimap) (STL/CLR)

List greater than or equal comparison.

## Syntax

```
template<typename Key,
         typename Mapped>
bool operator>=(multimap<Key, Mapped>% left,
                  multimap<Key, Mapped>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two multimaps are compared element by element.

### Example

```
// cliext_multimap_operator_ge.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

# multiset (STL/CLR)

9/20/2022 • 37 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `multiset` to manage a sequence of elements as a (nearly) balanced ordered tree of nodes, each storing one element.

In the description below, `GValue` is the same as `GKey`, which in turn is the same as `Key` unless the latter is a ref type, in which case it is `Key^`.

## Syntax

```
template<typename Key>
ref class multiset
    : public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
System::Collections::Generic::IList<GValue>,
Microsoft::VisualC::StlClr::ITree<Gkey, GValue>
{ ..... };
```

## Parameters

### Key

The type of the key component of an element in the controlled sequence.

## Requirements

**Header:** <cliext/set>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">multiset::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.
<a href="#">multiset::const_reference (STL/CLR)</a>	The type of a constant reference to an element.
<a href="#">multiset::const_reverse_iterator (STL/CLR)</a>	The type of a constant reverse iterator for the controlled sequence.
<a href="#">multiset::difference_type (STL/CLR)</a>	The type of a (possibly signed) distance between two elements.
<a href="#">multiset::generic_container (STL/CLR)</a>	The type of the generic interface for the container.

TYPE DEFINITION	DESCRIPTION
<a href="#">multiset::generic_iterator (STL/CLR)</a>	The type of an iterator for the generic interface for the container.
<a href="#">multiset::generic_reverse_iterator (STL/CLR)</a>	The type of a reverse iterator for the generic interface for the container.
<a href="#">multiset::generic_value (STL/CLR)</a>	The type of an element for the generic interface for the container.
<a href="#">multiset::iterator (STL/CLR)</a>	The type of an iterator for the controlled sequence.
<a href="#">multiset::key_compare (STL/CLR)</a>	The ordering delegate for two keys.
<a href="#">multiset::key_type (STL/CLR)</a>	The type of an ordering key.
<a href="#">multiset::reference (STL/CLR)</a>	The type of a reference to an element.
<a href="#">multiset::reverse_iterator (STL/CLR)</a>	The type of a reverse iterator for the controlled sequence.
<a href="#">multiset::size_type (STL/CLR)</a>	The type of a (non-negative) distance between two elements.
<a href="#">multiset::value_compare (STL/CLR)</a>	The ordering delegate for two element values.
<a href="#">multiset::value_type (STL/CLR)</a>	The type of an element.

MEMBER FUNCTION	DESCRIPTION
<a href="#">multiset::begin (STL/CLR)</a>	Designates the beginning of the controlled sequence.
<a href="#">multiset::clear (STL/CLR)</a>	Removes all elements.
<a href="#">multiset::count (STL/CLR)</a>	Counts elements matching a specified key.
<a href="#">multiset::empty (STL/CLR)</a>	Tests whether no elements are present.
<a href="#">multiset::end (STL/CLR)</a>	Designates the end of the controlled sequence.
<a href="#">multiset::equal_range (STL/CLR)</a>	Finds range that matches a specified key.
<a href="#">multiset::erase (STL/CLR)</a>	Removes elements at specified positions.
<a href="#">multiset::find (STL/CLR)</a>	Finds an element that matches a specified key.
<a href="#">multiset::insert (STL/CLR)</a>	Adds elements.
<a href="#">multiset::key_comp (STL/CLR)</a>	Copies the ordering delegate for two keys.
<a href="#">multiset::lower_bound (STL/CLR)</a>	Finds beginning of range that matches a specified key.
<a href="#">multiset::make_value (STL/CLR)</a>	Constructs a value object.

MEMBER FUNCTION	DESCRIPTION
<code>multiset::multiset (STL/CLR)</code>	Constructs a container object.
<code>multiset::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.
<code>multiset::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>multiset::size (STL/CLR)</code>	Counts the number of elements.
<code>multiset::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>multiset::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>multiset::upper_bound (STL/CLR)</code>	Finds end of range that matches a specified key.
<code>multiset::value_comp (STL/CLR)</code>	Copies the ordering delegate for two element values.
OPERATOR	DESCRIPTION
<code>multiset::operator= (STL/CLR)</code>	Replaces the controlled sequence.
<code>operator!= (multiset) (STL/CLR)</code>	Determines if a <code>multiset</code> object is not equal to another <code>multiset</code> object.
<code>operator&lt; (multiset) (STL/CLR)</code>	Determines if a <code>multiset</code> object is less than another <code>multiset</code> object.
<code>operator&lt;= (multiset) (STL/CLR)</code>	Determines if a <code>multiset</code> object is less than or equal to another <code>multiset</code> object.
<code>operator== (multiset) (STL/CLR)</code>	Determines if a <code>multiset</code> object is equal to another <code>multiset</code> object.
<code>operator&gt; (multiset) (STL/CLR)</code>	Determines if a <code>multiset</code> object is greater than another <code>multiset</code> object.
<code>operator&gt;= (multiset) (STL/CLR)</code>	Determines if a <code>multiset</code> object is greater than or equal to another <code>multiset</code> object.

## Interfaces

INTERFACE	DESCRIPTION
<code>ICloneable</code>	Duplicate an object.
<code>IEnumerable</code>	Sequence through elements.
<code>ICollection</code>	Maintain group of elements.

INTERFACE	DESCRIPTION
<code>IEnumerable&lt;T&gt;</code>	Sequence through typed elements.
<code>ICollection&lt;T&gt;</code>	Maintain group of typed elements.
<code>ITree&lt;Key, Value&gt;</code>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes. It inserts elements into a (nearly) balanced tree that it keeps ordered by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements.

The object orders the sequence it controls by calling a stored delegate object of type `multiset::key_compare (STL/CLR)`. You can specify the stored delegate object when you construct the multiset; if you specify no delegate object, the default is the comparison `operator<(key_type, key_type)`. You access this stored object by calling the member function `multiset::key_comp (STL/CLR)()`.

Such a delegate object must impose a strict weak ordering on keys of type `multiset::key_type (STL/CLR)`. That means, for any two keys `x` and `y`:

`key_comp()(x, y)` returns the same Boolean result on every call.

If `key_comp()(x, y)` is true, then `key_comp()(y, x)` must be false.

If `key_comp()(x, y)` is true, then `x` is said to be ordered before `y`.

If `!key_comp()(x, y) && !key_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

For any element `x` that precedes `y` in the controlled sequence, `key_comp()(y, x)` is false. (For the default delegate object, keys never decrease in value.) Unlike template class `set (STL/CLR)`, an object of template class `multiset` does not require that keys for all elements are unique. (Two or more keys can have equivalent ordering.)

Each element serves as both a key and a value. The sequence is represented in a way that permits lookup, insertion, and removal of an arbitrary element with a number of operations proportional to the logarithm of the number of elements in the sequence (logarithmic time). Moreover, inserting an element invalidates no iterators, and removing an element invalidates only those iterators which point at the removed element.

A multiset supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by `multiset::end (STL/CLR)()`. You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a multiset iterator to reach the head node, and it will then compare equal to `end()`. But you cannot dereference the iterator returned by `end()`.

Note that you cannot refer to a multiset element directly given its numerical position -- that requires a random-access iterator.

A multiset iterator stores a handle to its associated multiset node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A multiset iterator remains valid so long as its associated multiset node is associated with some multiset. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all

elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

## Members

### multiset::begin (STL/CLR)

Designates the beginning of the controlled sequence.

#### Syntax

```
iterator begin();
```

#### Remarks

The member function returns a bidirectional iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

#### Example

```
// cliext_multiset_begin.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Mymultiset::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("**begin() = {0}", *++it);
    return (0);
}
```

```
a b c
*begin() = a
**begin() = b
```

### multiset::clear (STL/CLR)

Removes all elements.

#### Syntax

```
void clear();
```

## Remarks

The member function effectively calls `multiset::erase (STL/CLR)` ( `multiset::begin (STL/CLR) ()`, `multiset::end (STL/CLR) ()` ). You use it to ensure that the controlled sequence is empty.

## Example

```
// cliext_multiset_clear.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
size() = 0
a b
size() = 0
```

## multiset::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

## Syntax

```
typedef T2 const_iterator;
```

## Remarks

The type describes an object of unspecified type `T2` that can serve as a constant bidirectional iterator for the controlled sequence.

## Example

```

// cliext_multiset_const_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Mymultiset::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## `multiset::const_reference` (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_multiset_const_reference.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Mymultiset::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Mymultiset::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

## multiset::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```
// cliext_multiset_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Mymultiset::const_reverse_iterator crit = c1.rbegin();
    for ( ; crit != c1.rend(); ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## multiset::count (STL/CLR)

Finds the number of elements matching a specified key.

### Syntax

```
size_type count(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns the number of elements in the controlled sequence that have equivalent ordering with *key*. You use it to determine the number of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_multiset_count.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
a b c
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

## multiset::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a possibly negative element count.

## Example

```

// cliext_multiset_difference_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mymultiset::difference_type diff = 0;
    for (Mymultiset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Mymultiset::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

## multiset::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

### Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [multiset::size \(STL/CLR\) \(\) == 0](#). You use it to test whether the multiset is empty.

### Example

```

// cliext_multiset_empty.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

## multiset::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a bidirectional iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_multiset_end.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    Mymultiset::iterator it = c1.end();
    --it;
    System::Console::WriteLine("--- --end() = {0}", *--it);
    System::Console::WriteLine("---end() = {0}", *++it);
    return (0);
}

```

```

a b c
*-- --end() = b
*--end() = c

```

## multiset::equal\_range (STL/CLR)

Finds range that matches a specified key.

### Syntax

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns a pair of iterators `cliext::pair<iterator, iterator>( multiset::lower_bound (STL/CLR) (key), multiset::upper_bound (STL/CLR) (key))`. You use it to determine the range of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_multiset_equal_range.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
typedef Mymultiset::pair_iter_iter Pairii;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("{0} ", *pair1.first);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
equal_range(L'x') empty = True
b

```

## multiset::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
size_type erase(key_type key)

```

### Parameters

*first*

Beginning of range to erase.

*key*

Key value to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*, and returns an

iterator that designates the first element remaining beyond the element removed, or [multiset::end \(STL/CLR\)](#) () if no such element exists. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [ `first` , `last` ), and returns an iterator that designates the first element remaining beyond any elements removed, or `end()` if no such element exists.. You use it to remove zero or more contiguous elements.

The third member function removes any element of the controlled sequence whose key has equivalent ordering to `key`, and returns a count of the number of elements removed. You use it to remove and count all elements that match a specified key.

Each element erasure takes time proportional to the logarithm of the number of elements in the controlled sequence.

## Example

```
// cliext_multiset_erase.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.insert(L'd');
    c1.insert(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    Mymultiset::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

## multiset::find (STL/CLR)

Finds an element that matches a specified key.

## Syntax

```
iterator find(key_type key);
```

## Parameters

### key

Key value to search for.

## Remarks

If at least one element in the controlled sequence has equivalent ordering with *key*, the member function returns an iterator designating one of those elements; otherwise it returns [multiset::end \(STL/CLR\)](#) (). You use it to locate an element currently in the controlled sequence that matches a specified key.

## Example

```
// cliext_multiset_find.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());
    System::Console::WriteLine("find {0} = {1}",
        L'b', *c1.find(L'b'));
    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
a b c
find A = False
find b = b
find C = False
```

## multiset::generic\_container (STL/CLR)

The type of the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::
    ITree<GKey, GValue>
generic_container;
```

## Remarks

The type describes the generic interface for this template container class.

## Example

```
// cliext_multiset_generic_container.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mymultiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(L'e');
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
a b c d
a b c d e
```

## multiset::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

## Example

```

// cliext_multiset_generic_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mymultiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Mymultiset::generic_iterator gcit = gc1->begin();
    Mymultiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

## multiset::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value>
generic_reverse_iterator;

```

### Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_multiset_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mymultiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Mymultiset::generic_reverse_iterator gcit = gc1->rbegin();
    Mymultiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
c

```

## multiset::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```

typedef GValue generic_value;

```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

### Example

```

// cliext_multiset_generic_value.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mymultiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Mymultiset::generic_iterator gcit = gc1->begin();
    Mymultiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

## multiset::insert (STL/CLR)

Adds elements.

### Syntax

```

iterator insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);

```

### Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Key value to insert.

*where*

Where in container to insert (hint only).

### Remarks

Each of the member functions inserts a sequence specified by the remaining operands.

The first member function inserts an element with value *val*, and returns an iterator that designates the newly inserted element. You use it to insert a single element.

The second member function inserts an element with value *val*, using *where* as a hint (to improve performance), and returns an iterator that designates the newly inserted element. You use it to insert a single element which might be adjacent to an element you know.

The third member function inserts the sequence [`first`, `last`). You use it to insert zero or more elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

Each element insertion takes time proportional to the logarithm of the number of elements in the controlled sequence. Insertion can occur in amortized constant time, however, given a hint that designates an element adjacent to the insertion point.

### Example

```

// cliext_multiset_insert.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    System::Console::WriteLine("insert(L'x') = {0}",
        *c1.insert(L'x'));

    System::Console::WriteLine("insert(L'b') = {0}",
        *c1.insert(L'b'));

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value with hint
    System::Console::WriteLine("insert(begin(), L'y') = {0}",
        *c1.insert(c1.begin(), L'y'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    Mymultiset c2;
    Mymultiset::iterator it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    Mymultiset c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)c1);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(L'x') = x
insert(L'b') = b
a b b c x
insert(begin(), L'y') = y
a b b c x y
a b b c x
a b b c x y

```

## multiset::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

## Syntax

```
typedef T1 iterator;
```

## Remarks

The type describes an object of unspecified type `T1` that can serve as a bidirectional iterator for the controlled sequence.

## Example

```
// cliext_multiset_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Mymultiset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## multiset::key\_comp (STL/CLR)

Copies the ordering delegate for two keys.

## Syntax

```
key_compare^key_comp();
```

## Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two keys.

## Example

```

// cliext_multiset_key_comp.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    Mymultiset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymultiset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## multiset::key\_compare (STL/CLR)

The ordering delegate for two keys.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its key arguments.

### Example

```

// cliext_multiset_key_compare.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    Mymultiset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymultiset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## multiset::key\_type (STL/CLR)

The type of an ordering key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter *Key*.

### Example

```

// cliext_multiset_key_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using key_type
    for (Mymultiset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Mymultiset::key_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## multiset::lower\_bound (STL/CLR)

Finds beginning of range that matches a specified key.

### Syntax

```
iterator lower_bound(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the first element  $x$  in the controlled sequence that has equivalent ordering to *key*. If no such element exists, it returns [multiset::end \(STL/CLR\)](#) () ; otherwise it returns an iterator that designates  $x$ . You use it to locate the beginning of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_multiset_lower_bound.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    System::Console::WriteLine("*lower_bound(L'a') = {0}",
        *c1.lower_bound(L'a'));
    System::Console::WriteLine("*lower_bound(L'b') = {0}",
        *c1.lower_bound(L'b'));
    return (0);
}

```

```

a b c
lower_bound(L'x') == end() = True
*lower_bound(L'a') = a
*lower_bound(L'b') = b

```

## multiset::make\_value (STL/CLR)

Constructs a value object.

### Syntax

```

static value_type make_value(key_type key);

```

### Parameters

*key*

Key value to use.

### Remarks

The member function returns a `value_type` object whose key is *key*. You use it to compose an object suitable for use with several other member functions.

### Example

```

// cliext_multiset_make_value.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(Mymultiset::make_value(L'a'));
    c1.insert(Mymultiset::make_value(L'b'));
    c1.insert(Mymultiset::make_value(L'c'));

    // display contents " a b c"
    for each (Mymultiset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## multiset::multiset (STL/CLR)

Constructs a container object.

### Syntax

```

multiset();
explicit multiset(key_compare^ pred);
multiset(multiset<Key>% right);
multiset(multiset<Key>^ right);
template<typename InIter>
    multiset(multiset<InIter first, InIter last>);
template<typename InIter>
    multiset(InIter first, InIter last,
            key_compare^ pred);
multiset(System::Collections::Generic::IEnumerable<GValue>^ right);
multiset(System::Collections::Generic::IEnumerable<GValue>^ right,
        key_compare^ pred);

```

### Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

### Remarks

The constructor:

`multiset();`

initializes the controlled sequence with no elements, with the default ordering predicate `key_compare()`. You use

it to specify an empty initial controlled sequence, with the default ordering predicate.

The constructor:

```
explicit multiset(key_compare^ pred);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate.

The constructor:

```
multiset(multiset<Key>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`], with the default ordering predicate. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the *multiset* object *right*, with the default ordering predicate.

The constructor:

```
multiset(multiset<Key>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`], with the default ordering predicate. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the *multiset* object *right*, with the default ordering predicate.

The constructor:

```
template<typename InIter> multiset(InIter first, InIter last);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the default ordering predicate. You use it to make the controlled sequence a copy of another sequence, with the default ordering predicate.

The constructor:

```
template<typename InIter> multiset(InIter first, InIter last, key_compare^ pred);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate.

The constructor:

```
multiset(System::Collections::Generic::IEnumerable<Key>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the default ordering predicate. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the default ordering predicate.

The constructor:

```
multiset(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate.

## Example

```
// cliext_multiset_construct.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
```

```

{
// construct an empty container
Mymultiset c1;
System::Console::WriteLine("size() = {0}", c1.size());

c1.insert(L'a');
c1.insert(L'b');
c1.insert(L'c');
for each (wchar_t elem in c1)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an ordering rule
Mymultiset c2 = cliext::greater_equal<wchar_t>();
System::Console::WriteLine("size() = {0}", c2.size());

c2.insert(c1.begin(), c1.end());
for each (wchar_t elem in c2)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range
Mymultiset c3(c1.begin(), c1.end());
for each (wchar_t elem in c3)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Mymultiset c4(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c4)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration
Mymultiset c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
for each (wchar_t elem in c5)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Mymultiset c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c6)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct from a generic container
Mymultiset c7(c4);
for each (wchar_t elem in c7)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct by copying another container
Mymultiset c8(%c3);
for each (wchar_t elem in c8)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
a b c
size() = 0
c b a
a b c
c b a
a b c
c b a
c b a
a b c
```

## multiset::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
multiset<Key>% operator=(multiset<Key>% right);
```

### Parameters

*right*

Container to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

### Example

```
// cliext_multiset_operator_as.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (Mymultiset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2 = c1;
    // display contents " a b c"
    for each (Mymultiset::value_type elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

## multiset::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

### Syntax

```
reverse_iterator rbegin();
```

### Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the **beginning** of the reverse sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```
// cliext_multiset_rbegin.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymultiset::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("**rbegin() = {0}", *++rit);
    return (0);
}
```

```
a b c
*rbegin() = c
*++rbegin() = b
```

## multiset::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_multiset_reference.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Mymultiset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        Mymultiset::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## multiset::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

### Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_multiset_rend.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Mymultiset::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("--- --rend() = {0}", *--rit);
    System::Console::WriteLine("---rend() = {0}", *++rit);
    return (0);
}

```

```

a b c
--- --rend() = b
---rend() = a

```

## multiset::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

### Syntax

```

typedef T3 reverse_iterator;

```

### Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

### Example

```
// cliext_multiset_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Mymultiset::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## multiset::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [multiset::empty \(STL/CLR\) \(\)](#).

### Example

```

// cliext_multiset_size.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

## multiset::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```

typedef int size_type;

```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_multiset_size_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mymultiset::size_type diff = 0;
    for (Mymultiset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3

```

## multiset::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```

void swap(multiset<Key>% right);

```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_multiset_swap.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Mymultiset c2;
    c2.insert(L'd');
    c2.insert(L'e');
    c2.insert(L'f');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
d e f
d e f
a b c

```

## multiset::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<value_type>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_multiset_to_array.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c d
a b c

```

## multiset::upper\_bound (STL/CLR)

Finds end of range that matches a specified key.

### Syntax

```

iterator upper_bound(key_type key);

```

### Parameters

#### *key*

Key value to search for.

### Remarks

The member function determines the last element *x* in the controlled sequence that has equivalent ordering to *key*. If no such element exists, or if *x* is the last element in the controlled sequence, it returns [multiset::end \(STL/CLR\)](#); otherwise it returns an iterator that designates the first element beyond *x*. You use it to locate the end of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_multiset_upper_bound.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    System::Console::WriteLine("*upper_bound(L'a') = {0}",
        *c1.upper_bound(L'a'));
    System::Console::WriteLine("*upper_bound(L'b') = {0}",
        *c1.upper_bound(L'b'));
    return (0);
}

```

```

a b c
upper_bound(L'x') == end() = True
*upper_bound(L'a') = b
*upper_bound(L'b') = c

```

## multiset::value\_comp (STL/CLR)

Copies the ordering delegate for two element values.

### Syntax

```

value_compare^ value_comp();

```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two element values.

### Example

```

// cliext_multiset_value_comp.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    Mymultiset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

```

## multiset::value\_compare (STL/CLR)

The ordering delegate for two element values.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its value arguments.

### Example

```

// cliext_multiset_value_compare.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    Mymultiset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```
compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

## multiset::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef generic_value value_type;
```

### Remarks

The type is a synonym for `generic_value`.

### Example

```
// cliext_multiset_value_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using value_type
    for (Mymultiset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Mymultiset::value_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## operator!= (multiset) (STL/CLR)

List not equal comparison.

### Syntax

```
template<typename Key>
bool operator!=(multiset<Key>% left,
                 multiset<Key>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two multisets are compared element by element.

## Example

```
// cliext_multiset_operator_ne.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}
```

```
a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True
```

## operator< (multiset) (STL/CLR)

List less than comparison.

## Syntax

```
template<typename Key>
bool operator<(multiset<Key>% left,
    multiset<Key>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns true if, for the lowest position `i` for which `!(right[i] < left[i])` it is also true that `left[i] < right[i]`. Otherwise, it returns `left->size() < right->size()`. You use it to test whether *left* is ordered before *right* when the two multisets are compared element by element.

## Example

```
// cliext_multiset_operator_lt.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

## operator<= (multiset) (STL/CLR)

List less than or equal comparison.

## Syntax

```
template<typename Key>
bool operator<=(multiset<Key>% left,
                 multiset<Key>% right);
```

#### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

#### Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two multisets are compared element by element.

#### Example

```
// cliext_multiset_operator_le.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
                           c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
                           c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator== (multiset) (STL/CLR)

List equal comparison.

## Syntax

```
template<typename Key>
bool operator==(multiset<Key>% left,
                  multiset<Key>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position *i*, *left[i] == right[i]*. You use it to test whether *left* is ordered the same as *right* when the two multisets are compared element by element.

### Example

```
// cliext_multiset_operator_eq.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
                             c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
                             c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

## operator> (multiset) (STL/CLR)

List greater than comparison.

## Syntax

```
template<typename Key>
    bool operator>(multiset<Key>% left,
                      multiset<Key>% right);
```

## Parameters

/eft

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two multisets are compared element by element.

## Example

```
// cliext_multiset_operator_gt.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

## operator>= (multiset) (STL/CLR)

List greater than or equal comparison.

### Syntax

```
template<typename Key>
bool operator>=(multiset<Key>% left,
                 multiset<Key>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two multisets are compared element by element.

### Example

```
// cliext_multiset_operator_ge.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
        c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

# numeric (STL/CLR)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Defines container template functions that perform algorithms provided for numerical processing.

## Syntax

```
#include <cliext/numeric>
```

## Requirements

**Header:** <cliext/numeric>

**Namespace:** cliext

## Declarations

FUNCTION	DESCRIPTION
<a href="#">accumulate (STL/CLR)</a>	Computes the sum of all the elements in a specified range including some initial value by computing successive partial sums or computes the result of successive partial results similarly obtained from using a specified binary operation other than the sum.
<a href="#">adjacent_difference (STL/CLR)</a>	Computes the successive differences between each element and its predecessor in an input range and outputs the results to a destination range or computes the result of a generalized procedure where the difference operation is replaced by another, specified binary operation.
<a href="#">inner_product (STL/CLR)</a>	Computes the sum of the element-wise product of two ranges and adds it to a specified initial value or computes the result of a generalized procedure where the sum and product binary operations are replaced by other specified binary operations.
<a href="#">partial_sum (STL/CLR)</a>	Computes a series of sums in an input range from the first element through the <code>i</code> th element and stores the result of each such sum in <code>i</code> th element of a destination range or computes the result of a generalized procedure where the sum operation is replaced by another specified binary operation.

## Members

### [accumulate \(STL/CLR\)](#)

Computes the sum of all the elements in a specified range including some initial value by computing successive partial sums or computes the result of successive partial results similarly obtained from using a specified binary operation other than the sum.

## Syntax

```
template<class _InIt, class _Ty> inline
    _Ty accumulate(_InIt _First, _InIt _Last, _Ty _Val);
template<class _InIt, class _Ty, class _Fn2> inline
    _Ty accumulate(_InIt _First, _InIt _Last, _Ty _Val, _Fn2 _Func);
```

## Remarks

This function behaves the same as the C++ Standard Library numeric function `accumulate`. For more information, see [accumulate](#).

## adjacent\_difference (STL/CLR)

Computes the successive differences between each element and its predecessor in an input range and outputs the results to a destination range or computes the result of a generalized procedure where the difference operation is replaced by another, specified binary operation.

## Syntax

```
template<class _InIt, class _OutIt> inline
    _OutIt adjacent_difference(_InIt _First, _InIt _Last,
        _OutIt _Dest);
template<class _InIt, class _OutIt, class _Fn2> inline
    _OutIt adjacent_difference(_InIt _First, _InIt _Last,
        _OutIt _Dest, _Fn2 _Func);
```

## Remarks

This function behaves the same as the C++ Standard Library numeric function `adjacent_difference`. For more information, see [adjacent\\_difference](#).

## inner\_product (STL/CLR)

Computes the sum of the element-wise product of two ranges and adds it to a specified initial value or computes the result of a generalized procedure where the sum and product binary operations are replaced by other specified binary operations.

## Syntax

```
template<class _InIt1, class _InIt2, class _Ty> inline
    _Ty inner_product(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
        _Ty _Val);
template<class _InIt1, class _InIt2, class _Ty, class _Fn21,
        class _Fn22> inline
    _Ty inner_product(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
        _Ty _Val, _Fn21 _Func1, _Fn22 _Func2);
```

## Remarks

This function behaves the same as the C++ Standard Library numeric function `inner_product`. For more information, see [inner\\_product](#).

## partial\_sum (STL/CLR)

Computes a series of sums in an input range from the first element through the  $i$ th element and stores the result of each such sum in  $i$ th element of a destination range or computes the result of a generalized procedure where the sum operation is replaced by another specified binary operation.

## Syntax

```
template<class _InIt, class _OutIt> inline
    _OutIt partial_sum(_InIt _First, _InIt _Last, _OutIt _Dest);
template<class _InIt, class _OutIt, class _Fn2> inline
    _OutIt partial_sum(_InIt _First, _InIt _Last,
        _OutIt _Dest, _Fn2 _Func);
```

## Remarks

This function behaves the same as the C++ Standard Library numeric function `partial_sum`. For more information, see [partial\\_sum](#).

# priority\_queue (STL/CLR)

9/20/2022 • 17 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length ordered sequence of elements that has limited access. You use the container adapter `priority_queue` to manage an underlying container as a priority queue.

In the description below, `GValue` is the same as *Value* unless the latter is a ref type, in which case it is `Value^`. Similarly, `GContainer` is the same as *Container* unless the latter is a ref type, in which case it is `Container^`.

## Syntax

```
template<typename Value,
         typename Container>
ref class priority_queue
    System::ICloneable,
    Microsoft::VisualC::StlClr::IPriorityQueue<GValue, GContainer>
{ .... };
```

### Parameters

#### *Value*

The type of an element in the controlled sequence.

#### *Container*

The type of the underlying container.

## Requirements

**Header:** <cliext/queue>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<code>priority_queue::const_reference (STL/CLR)</code>	The type of a constant reference to an element.
<code>priority_queue::container_type (STL/CLR)</code>	The type of the underlying container.
<code>priority_queue::difference_type (STL/CLR)</code>	The type of a signed distance between two elements.
<code>priority_queue::generic_container (STL/CLR)</code>	The type of the generic interface for the container adapter.
<code>priority_queue::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container adapter.
<code>priority_queue::reference (STL/CLR)</code>	The type of a reference to an element.
<code>priority_queue::size_type (STL/CLR)</code>	The type of a signed distance between two elements.

TYPE DEFINITION	DESCRIPTION
<code>priority_queue::value_compare (STL/CLR)</code>	The ordering delegate for two elements.
<code>priority_queue::value_type (STL/CLR)</code>	The type of an element.
MEMBER FUNCTION	DESCRIPTION
<code>priority_queue::assign (STL/CLR)</code>	Replaces all elements.
<code>priority_queue::empty (STL/CLR)</code>	Tests whether no elements are present.
<code>priority_queue::get_container (STL/CLR)</code>	Accesses the underlying container.
<code>priority_queue::pop (STL/CLR)</code>	Removes the highest-priority element.
<code>priority_queue::priority_queue (STL/CLR)</code>	Constructs a container object.
<code>priority_queue::push (STL/CLR)</code>	Adds a new element.
<code>priority_queue::size (STL/CLR)</code>	Counts the number of elements.
<code>priority_queue::top (STL/CLR)</code>	Accesses the highest-priority element.
<code>priority_queue::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>priority_queue::value_comp (STL/CLR)</code>	Copies the ordering delegate for two elements.
PROPERTY	DESCRIPTION
<code>priority_queue::top_item (STL/CLR)</code>	Accesses the highest-priority element.
OPERATOR	DESCRIPTION
<code>priority_queue::operator= (STL/CLR)</code>	Replaces the controlled sequence.

## Interfaces

INTERFACE	DESCRIPTION
<code>ICloneable</code>	Duplicate an object.
<code>IPriorityQueue&lt;Value, Container&gt;</code>	Maintain generic container adapter.

## Remarks

The object allocates and frees storage for the sequence it controls through an underlying container, of type `Container`, that stores `Value` elements and grows on demand. It keeps the sequence ordered as a heap, with the highest-priority element (the top element) readily accessible and removable. The object restricts access to pushing new elements and popping just the highest-priority element, implementing a priority queue.

The object orders the sequence it controls by calling a stored delegate object of type [priority\\_queue::value\\_compare \(STL/CLR\)](#). You can specify the stored delegate object when you construct the `priority_queue`; if you specify no delegate object, the default is the comparison `operator<(value_type, value_type)`. You access this stored object by calling the member function [priority\\_queue::value\\_comp \(STL/CLR\)\(\)](#).

Such a delegate object must impose a strict weak ordering on values of type [priority\\_queue::value\\_type \(STL/CLR\)](#). That means, for any two keys `x` and `y`:

`value_comp()(x, y)` returns the same Boolean result on every call.

If `value_comp()(x, y)` is true, then `value_comp()(y, x)` must be false.

If `value_comp()(x, y)` is true, then `x` is said to be ordered before `y`.

If `!value_comp()(x, y) && !value_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

For any element `x` that precedes `y` in the controlled sequence, `key_comp()(y, x)` is false. (For the default delegate object, keys never decrease in value.)

The highest priority element is thus one of the elements which is not ordered before any other element.

Since the underlying container keeps elements ordered as a heap:

The container must support random-access iterators.

Elements with equivalent ordering may be popped in a different order than they were pushed. (The ordering is not stable.)

Thus, candidates for the underlying container include [deque \(STL/CLR\)](#) and [vector \(STL/CLR\)](#).

## Members

### `priority_queue::assign (STL/CLR)`

Replaces all elements.

#### Syntax

```
void assign(priority_queue<Value, Container>% right);
```

#### Parameters

`right`

Container adapter to insert.

#### Remarks

The member function assigns `right.get_container()` to the underlying container. You use it to change the entire contents of the queue.

#### Example

```

// cliext_priority_queue_assign.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign a repetition of values
    Mypriority_queue c2;
    c2.assign(c1);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

c a b
c a b

```

## priority\_queue::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```

typedef value_type% const_reference;

```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_priority_queue_const_reference.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " c b a"
    for ( ; !c1.empty(); c1.pop())
        { // get a const reference to an element
        Mypriority_queue::const_reference cref = c1.top();
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

c b a

## priority\_queue::container\_type (STL/CLR)

The type of the underlying container.

### Syntax

```
typedef Container value_type;
```

### Remarks

The type is a synonym for the template parameter `Container`.

### Example

```

// cliext_priority_queue_container_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c" using container_type
    Mypriority_queue::container_type wc1 = c1.get_container();
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

c a b

## priority\_queue::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a possibly negative element count.

### Example

```
// cliext_priority_queue_difference_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute negative difference
    Mypriority_queue::difference_type diff = c1.size();
    c1.push(L'd');
    c1.push(L'e');
    diff -= c1.size();
    System::Console::WriteLine("pushing 2 = {0}", diff);

    // compute positive difference
    diff = c1.size();
    c1.pop();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("popping 3 = {0}", diff);
    return (0);
}
```

```
c a b
pushing 2 = -2
popping 3 = 3
```

## priority\_queue::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

## Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [priority\\_queue::size \(STL/CLR\)](#) `() == 0`. You use it to test whether the priority\_queue is empty.

## Example

```
// cliext_priority_queue_empty.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.pop();
    c1.pop();
    c1.pop();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
c a b
size() = 3
empty() = False
size() = 0
empty() = True
```

## priority\_queue::generic\_container (STL/CLR)

The type of the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::IPriorityQueue<Value>
    generic_container;
```

## Remarks

The type describes the generic interface for this template container adapter class.

## Example

```

// cliext_priority_queue_generic_container.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mypriority_queue::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push(L'e');
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

c a b
c a b
d c b a
e d b a c

```

## priority\_queue::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```
typedef GValue generic_value;
```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class. (`GValue` is either `value_type` or `value_type^` if `value_type` is a ref type.)

### Example

```

// cliext_priority_queue_generic_value.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get interface to container
    Mypriority_queue::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display in priority order using generic_value
    for ( ; !gc1->empty(); gc1->pop())
    {
        Mypriority_queue::generic_value elem = gc1->top();

        System::Console::Write("{0} ", elem);
    }
    System::Console::WriteLine();
    return (0);
}

```

```

c a b
c a b
c b a

```

## priority\_queue::get\_container (STL/CLR)

Accesses the underlying container.

### Syntax

```

container_type get_container();

```

### Remarks

The member function returns the underlying container. You use it to bypass the restrictions imposed by the container wrapper.

### Example

```

// cliext_priority_queue_get_container.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

c a b

## priority\_queue::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
priority_queue <Value, Container>% operator=(priority_queue <Value, Container>% right);
```

### Parameters

*right*

Container adapter to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

### Example

```
// cliext_priority_queue_operator_as.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mypriority_queue c2;
    c2 = c1;
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c a b
c a b
```

## priority\_queue::pop (STL/CLR)

Removes the highest-priority element.

### Syntax

```
void pop();
```

### Remarks

The member function removes the highest-priority element of the controlled sequence, which must be non-empty. You use it to shorten the queue by one element at the back.

### Example

```

// cliext_priority_queue_pop.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop();
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

c a b
b a

```

## priority\_queue::priority\_queue (STL/CLR)

Constructs a container adapter object.

### Syntax

```

priority_queue();
priority_queue(priority_queue<Value, Container> right);
priority_queue(priority_queue<Value, Container> right);
explicit priority_queue(value_compare^ pred);
priority_queue(value_compare^ pred, container_type% cont);
template<typename InIt>
    priority_queue(InIt first, InIt last);
template<typename InIt>
    priority_queue(InIt first, InIt last,
                  value_compare^ pred);
template<typename InIt>
    priority_queue(InIt first, InIt last,
                  value_compare^ pred, container_type% cont);

```

### Parameters

*cont*

Container to copy.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

## Remarks

The constructor:

```
priority_queue();
```

creates an empty wrapped container, with the default ordering predicate. You use it to specify an empty initial controlled sequence, with the default ordering predicate.

The constructor:

```
priority_queue(priority_queue<Value, Container>% right);
```

creates a wrapped container that is a copy of `right.get_container()`, with the ordering predicate `right.value_comp()`. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the queue object *right*, with the same ordering predicate.

The constructor:

```
priority_queue(priority_queue<Value, Container>^ right);
```

creates a wrapped container that is a copy of `right->get_container()`, with the ordering predicate `right->value_comp()`. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the queue object `*right`, with the same ordering predicate.

The constructor:

```
explicit priority_queue(value_compare^ pred);
```

creates an empty wrapped container, with the ordering predicate *pred*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate.

The constructor:

```
priority_queue(value_compare^ pred, container_type cont);
```

creates an empty wrapped container, with the ordering predicate *pred*, then pushes all the elements of *cont*. You use it to specify an initial controlled sequence from an existing container, with the specified ordering predicate.

The constructor:

```
template<typename InIt> priority_queue(InIt first, InIt last);
```

creates an empty wrapped container, with the default ordering predicate, then pushes the sequence [`first`, `last`). You use it to specify an initial controlled sequence from a specified equeunce, with the specified ordering predicate.

The constructor:

```
template<typename InIt> priority_queue(InIt first, InIt last, value_compare^ pred);
```

creates an empty wrapped container, with the ordering predicate *pred*, then pushes the sequence [`first`, `last`). You use it to specify an initial controlled sequence from a specified sequeunce, with the specified ordering predicate.

The constructor:

```
template<typename InIt> priority_queue(InIt first, InIt last, value_compare^ pred, container_type% cont);
```

creates an empty wrapped container, with the ordering predicate *pred*, then pushes all the elements of *cont* plus

the sequence [ `first` , `last` ). You use it to specify an initial controlled sequence from an existing container and a specified sequence, with the specified ordering predicate.

## Example

```
// cliext_priority_queue_construct.cpp
// compile with: /clr
#include <cliext/queue>
#include <cliext/deque>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
typedef cliext::deque<wchar_t> Mydeque;
int main()
{
// construct an empty container
    Mypriority_queue c1;
    Mypriority_queue::container_type^ wc1 = c1.get_container();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an ordering rule
    Mypriority_queue c2 = cliext::greater<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    for each (wchar_t elem in wc1)
        c2.push(elem);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an ordering rule by copying an underlying container
    Mypriority_queue c2x =
        gcnew Mypriority_queue(cliext::greater<wchar_t>(), *wc1);
    for each (wchar_t elem in c2x.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range
    Mypriority_queue c3(wc1->begin(), wc1->end());
    for each (wchar_t elem in c3.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range and an ordering rule
    Mypriority_queue c4(wc1->begin(), wc1->end(),
        cliext::greater<wchar_t>());
    for each (wchar_t elem in c4.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range, another container, and an ordering rule
    Mypriority_queue c5(wc1->begin(), wc1->end(),
        cliext::greater<wchar_t>(), *wc1);
    for each (wchar_t elem in c5.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct from a generic container
    Mypriority_queue c6(c3);
    for each (wchar_t elem in c6.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
```

```

// construct by copying another container
Mypriority_queue c7(%c3);
for each (wchar_t elem in c7.get_container())
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an ordering rule, by copying an underlying container
Mypriority_queue c8 =
    gcnew Mypriority_queue(cliext::greater<wchar_t>(), *wc1);
for each (wchar_t elem in c8.get_container())
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

return (0);
}

```

```

size() = 0
c a b
size() = 0
a c b
a c b
c a b
a c b
a a b c c b
c a b
c a b
a c b

```

## priority\_queue::push (STL/CLR)

Adds a new element.

### Syntax

```
void push(value_type val);
```

### Remarks

The member function inserts an element with value `val` into the controlled sequence, and reorders the controlled sequence to maintain the heap discipline. You use it to add another element to the queue.

### Example

```

// cliext_priority_queue_push.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

c a b

## priority\_queue::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_priority_queue_reference.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify top of priority_queue and redisplay
    Mypriority_queue::reference ref = c1.top();
    ref = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
c a b  
x a b
```

## priority\_queue::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [priority\\_queue::empty \(STL/CLR\) \(\)](#).

### Example

```
// cliext_priority_queue_size.cpp  
// compile with: /clr  
#include <cliext/queue>  
  
typedef cliext::priority_queue<wchar_t> Mypriority_queue;  
int main()  
{  
    Mypriority_queue c1;  
    c1.push(L'a');  
    c1.push(L'b');  
    c1.push(L'c');  
  
    // display initial contents " a b c"  
    for each (wchar_t elem in c1.get_container())  
        System::Console::Write("{0} ", elem);  
    System::Console::WriteLine();  
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());  
  
    // pop an item and reinspect  
    c1.pop();  
    System::Console::WriteLine("size() = {0} after popping", c1.size());  
  
    // add two elements and reinspect  
    c1.push(L'a');  
    c1.push(L'b');  
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());  
    return (0);  
}
```

```
c a b  
size() = 3 starting with 3  
size() = 2 after popping  
size() = 4 after adding 2
```

## priority\_queue::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

## Remarks

The type describes a non-negative element count.

## Example

```
// cliext_priority_queue_size_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mypriority_queue::size_type diff = c1.size();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("size difference = {0}", diff);
    return (0);
}
```

```
c a b
size difference = 2
```

## priority\_queue::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

## Syntax

```
cli::array<Value>^ to_array();
```

## Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

## Example

```

// cliext_priority_queue_to_array.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

d c b a
c a b

```

## priority\_queue::top (STL/CLR)

Accesses the highest-priority element.

### Syntax

```

reference top();

```

### Remarks

The member function returns a reference to the top (highest-priority) element of the controlled sequence, which must be non-empty. You use it to access the highest-priority element, when you know it exists.

### Example

```

// cliext_priority_queue_top.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("top() = {0}", c1.top());

    // alter last item and reinspect
    c1.top() = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

## priority\_queue::top\_item (STL/CLR)

Accesses the highest-priority element.

### Syntax

```
property value_type back_item;
```

### Remarks

The property accesses the top (highest-priority) element of the controlled sequence, which must be non-empty. You use it to read or write the highest-priority element, when you know it exists.

### Example

```

// cliext_priority_queue_top_item.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("top_item = {0}", c1.top_item);

    // alter last item and reinspect
    c1.top_item = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

c a b
top_item = c
x a b

```

## priority\_queue::value\_comp (STL/CLR)

Copies the ordering delegate for two elements.

### Syntax

```

value_compare^ value_comp();

```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two values.

### Example

```

// cliext_priority_queue_value_comp.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    Mypriority_queue::value_compare^ vcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        vcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        vcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        vcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mypriority_queue c2 = cliext::greater<wchar_t>();
    vcomp = c2.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        vcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        vcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        vcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## priority\_queue::value\_compare (STL/CLR)

The ordering delegate for two values.

### Syntax

```
binary_delegate<value_type, value_type, int> value_compare;
```

### Remarks

The type is a synonym for the delegate that determines whether the first argument is ordered before the second.

### Example

```

// cliext_priority_queue_value_compare.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    Mypriority_queue::value_compare^ vcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        vcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        vcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        vcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mypriority_queue c2 = cliext::greater<wchar_t>();
    vcomp = c2.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        vcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        vcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        vcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## priority\_queue::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef Value value_type;
```

### Remarks

The type is a synonym for the template parameter *Value*.

### Example

```
// cliext_priority_queue_value_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " a b c" using value_type
    for ( ; !c1.empty(); c1.pop())
        {   // store element in value_type object
            Mypriority_queue::value_type val = c1.top();

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

# queue (STL/CLR)

9/20/2022 • 19 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has first-in first-out access. You use the container adapter `queue` to manage an underlying container as a queue.

In the description below, `GValue` is the same as *Value* unless the latter is a ref type, in which case it is `Value^`. Similarly, `GContainer` is the same as *Container* unless the latter is a ref type, in which case it is `Container^`.

## Syntax

```
template<typename Value,
         typename Container>
ref class queue
    : public
        System::ICloneable,
        Microsoft::VisualC::StlClr::IQueue<GValue, GContainer>
{ .... };
```

### Parameters

#### *Value*

The type of an element in the controlled sequence.

#### *Container*

The type of the underlying container.

## Requirements

**Header:** <cliext/queue>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<code>queue::const_reference (STL/CLR)</code>	The type of a constant reference to an element.
<code>queue::container_type (STL/CLR)</code>	The type of the underlying container.
<code>queue::difference_type (STL/CLR)</code>	The type of a signed distance between two elements.
<code>queue::generic_container (STL/CLR)</code>	The type of the generic interface for the container adapter.
<code>queue::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container adapter.
<code>queue::reference (STL/CLR)</code>	The type of a reference to an element.
<code>queue::size_type (STL/CLR)</code>	The type of a signed distance between two elements.

TYPE DEFINITION	DESCRIPTION
queue::value_type (STL/CLR)	The type of an element.
MEMBER FUNCTION	DESCRIPTION
queue::assign (STL/CLR)	Replaces all elements.
queue::back (STL/CLR)	Accesses the last element.
queue::empty (STL/CLR)	Tests whether no elements are present.
queue::front (STL/CLR)	Accesses the first element.
queue::get_container (STL/CLR)	Accesses the underlying container.
queue::pop (STL/CLR)	Removes the first element.
queue::push (STL/CLR)	Adds a new last element.
queue::queue (STL/CLR)	Constructs a container object.
queue::size (STL/CLR)	Counts the number of elements.
queue::to_array (STL/CLR)	Copies the controlled sequence to a new array.
PROPERTY	DESCRIPTION
queue::back_item (STL/CLR)	Accesses the last element.
queue::front_item (STL/CLR)	Accesses the first element.
OPERATOR	DESCRIPTION
queue::operator= (STL/CLR)	Replaces the controlled sequence.
operator!= (queue) (STL/CLR)	Determines if a <code>queue</code> object is not equal to another <code>queue</code> object.
operator< (queue) (STL/CLR)	Determines if a <code>queue</code> object is less than another <code>queue</code> object.
operator<= (queue) (STL/CLR)	Determines if a <code>queue</code> object is less than or equal to another <code>queue</code> object.
operator== (queue) (STL/CLR)	Determines if a <code>queue</code> object is equal to another <code>queue</code> object.
operator> (queue) (STL/CLR)	Determines if a <code>queue</code> object is greater than another <code>queue</code> object.

OPERATOR	DESCRIPTION
<code>operator&gt;= (queue) (STL/CLR)</code>	Determines if a <code>queue</code> object is greater than or equal to another <code>queue</code> object.

## Interfaces

INTERFACE	DESCRIPTION
<code>ICloneable</code>	Duplicate an object.
<code>IQueue&lt;Value, Container&gt;</code>	Maintain generic container adapter.

## Remarks

The object allocates and frees storage for the sequence it controls through an underlying container, of type `Container`, that stores `Value` elements and grows on demand. The object restricts access to just pushing the first element and popping the last element, implementing a first-in first-out queue (also known as a FIFO queue, or simply a queue).

## Members

### queue::assign (STL/CLR)

Replaces all elements.

#### Syntax

```
void assign(queue<Value, Container>% right);
```

#### Parameters

*right*

Container adapter to insert.

#### Remarks

The member function assigns `right.get_container()` to the underlying container. You use it to change the entire contents of the queue.

#### Example

```

// cliext_queue_assign.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign a repetition of values
    Myqueue c2;
    c2.assign(c1);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

## queue::back (STL/CLR)

Accesses the last element.

### Syntax

```
reference back();
```

### Remarks

The member function returns a reference to the last element of the controlled sequence, which must be non-empty. You use it to access the last element, when you know it exists.

### Example

```

// cliext_queue_back.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back() = {0}", c1.back());

    // alter last item and reinspect
    c1.back() = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
back() = c
a b x

```

## queue::back\_item (STL/CLR)

Accesses the last element.

### Syntax

```
property value_type back_item;
```

### Remarks

The property accesses the last element of the controlled sequence, which must be non-empty. You use it to read or write the last element, when you know it exists.

### Example

```

// cliext_queue_back_item.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back_item = {0}", c1.back_item);

    // alter last item and reinspect
    c1.back_item = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
back_item = c
a b x

```

## queue::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```

typedef value_type% const_reference;

```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_queue_const_reference.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for ( ; !c1.empty(); c1.pop())
        { // get a const reference to an element
        Myqueue::const_reference cref = c1.front();
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## queue::container\_type (STL/CLR)

The type of the underlying container.

### Syntax

```
typedef Container value_type;
```

### Remarks

The type is a synonym for the template parameter `Container`.

### Example

```

// cliext_queue_container_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c" using container_type
    Myqueue::container_type wc1 = c1.get_container();
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## queue::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a possibly negative element count.

### Example

```
// cliext_queue_difference_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute negative difference
    Myqueue::difference_type diff = c1.size();
    c1.push(L'd');
    c1.push(L'e');
    diff -= c1.size();
    System::Console::WriteLine("pushing 2 = {0}", diff);

    // compute positive difference
    diff = c1.size();
    c1.pop();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("popping 3 = {0}", diff);
    return (0);
}
```

```
a b c
pushing 2 = -2
popping 3 = 3
```

## queue::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

## Remarks

The member function returns true for an empty controlled sequence. It is equivalent to `queue::size (STL/CLR)`  
`() == 0`. You use it to test whether the queue is empty.

## Example

```
// cliext_queue_empty.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.pop();
    c1.pop();
    c1.pop();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
a b c
size() = 3
empty() = False
size() = 0
empty() = True
```

## queue::front (STL/CLR)

Accesses the first element.

## Syntax

```
reference front();
```

## Remarks

The member function returns a reference to the first element of the controlled sequence, which must be non-empty. You use it to access the first element, when you know it exists.

## Example

```

// cliext_queue_front.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front() = {0}", c1.front());

    // alter first item and reinspect
    c1.front() = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
front() = a
x b c

```

## queue::front\_item (STL/CLR)

Accesses the first element.

### Syntax

```
property value_type front_item;
```

### Remarks

The property accesses the first element of the controlled sequence, which must be non-empty. You use it to read or write the first element, when you know it exists.

### Example

```

// cliext_queue_front_item.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("front_item = {0}", c1.front_item);

    // alter last item and reinspect
    c1.front_item = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
front_item = a
x b c

```

## queue::generic\_container (STL/CLR)

The type of the generic interface for the container adapter.

### Syntax

```

typedef Microsoft::VisualC::StlClr::IQueue<Value>
generic_container;

```

### Remarks

The type describes the generic interface for this template container adapter class.

### Example

```

// cliext_queue_generic_container.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myqueue::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push(L'e');
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

## queue::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```

typedef GValue generic_value;

```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class. (`GValue` is either `value_type` or `value_type^` if `value_type` is a ref type.)

### Example

```

// cliext_queue_generic_value.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get interface to container
    Myqueue::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display in order using generic_value
    for (; !gc1->empty(); gc1->pop())
    {
        Myqueue::generic_value elem = gc1->front();

        System::Console::Write("{0} ", elem);
    }
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c

```

## queue::get\_container (STL/CLR)

Accesses the underlying container.

### Syntax

```

container_type^ get_container();

```

### Remarks

The member function returns the underlying container. You use it to bypass the restrictions imposed by the container wrapper.

### Example

```

// cliext_queue_get_container.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## queue::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
queue <Value, Container>% operator=(queue <Value, Container>% right);
```

### Parameters

*right*

Container adapter to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

### Example

```

// cliext_queue_operator_as.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2 = c1;
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

## queue::pop (STL/CLR)

Removes the last element.

### Syntax

```

void pop();

```

### Remarks

The member function removes the last element of the controlled sequence, which must be non-empty. You use it to shorten the queue by one element at the back.

### Example

```

// cliext_queue_pop.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop();
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
b c

```

## queue::push (STL/CLR)

Adds a new last element.

### Syntax

```

void push(value_type val);

```

### Remarks

The member function adds an element with value `val` at the end of the queue. You use it to append an element to the queue.

### Example

```

// cliext_queue_push.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

## queue::queue (STL/CLR)

Constructs a container adapter object.

### Syntax

```

queue();
queue(queue<Value, Container>% right);
queue(queue<Value, Container>^ right);
explicit queue(container_type% wrapped);

```

### Parameters

*right*

Object to copy.

*wrapped*

Wrapped container to use.

### Remarks

The constructor:

```
queue();
```

creates an empty wrapped container. You use it to specify an empty initial controlled sequence.

The constructor:

```
queue(queue<Value, Container>% right);
```

creates a wrapped container that is a copy of `right.get_container()`. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the queue object *right*.

The constructor:

```
queue(queue<Value, Container>^ right);
```

creates a wrapped container that is a copy of `right->get_container()`. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the queue object `*right`.

The constructor:

```
explicit queue(container_type wrapped);
```

uses the existing container *wrapped* as the wrapped container. You use it to construct a queue from an existing container.

## Example

```
// cliext_queue_construct.cpp
// compile with: /clr
#include <cliext/queue>
#include <cliext/list>

typedef cliext::queue<wchar_t> Myqueue;
typedef cliext::list<wchar_t> Mylist;
typedef cliext::queue<wchar_t, Mylist> Myqueue_list;
int main()
{
// construct an empty container
    Myqueue c1;
    System::Console::WriteLine("size() = {0}", c1.size());

// construct from an underlying container
    Mylist v2(5, L'x');
    Myqueue_list c2(v2);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying another container
    Myqueue_list c3(c2);
    for each (wchar_t elem in c3.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying another container through handle
    Myqueue_list c4(%c2);
    for each (wchar_t elem in c4.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
size() = 0
x x x x x
x x x x x
x x x x x
```

## queue::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

## Example

```

// cliext_queue_reference.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify back of queue and redisplay
    Myqueue::reference ref = c1.back();
    ref = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b x

```

## queue::size (STL/CLR)

Counts the number of elements.

### Syntax

```

size_type size();

```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [queue::empty \(STL/CLR\) \(\)](#).

### Example

```

// cliext_queue_size.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // pop an item and reinspect
    c1.pop();
    System::Console::WriteLine("size() = {0} after popping", c1.size());

    // add two elements and reinspect
    c1.push(L'a');
    c1.push(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 2 after popping
size() = 4 after adding 2

```

## queue::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_queue_size_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myqueue::size_type diff = c1.size();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("size difference = {0}", diff);
    return (0);
}

```

```

a b c
size difference = 2

```

## queue::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<Value>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_queue_to_array.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c d
a b c

```

## queue::value\_type (STL/CLR)

The type of an element.

### Syntax

```

typedef Value value_type;

```

### Remarks

The type is a synonym for the template parameter *Value*.

### Example

```

// cliext_queue_value_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " a b c" using value_type
    for ( ; !c1.empty(); c1.pop())
        {   // store element in value_type object
            Myqueue::value_type val = c1.front();

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## operator!= (queue) (STL/CLR)

Queue not equal comparison.

### Syntax

```

template<typename Value,
         typename Container>
bool operator!=(queue<Value, Container>% left,
                  queue<Value, Container>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two queues are compared element by element.

### Example

```

// cliext_queue_operator_ne.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

## operator< (queue) (STL/CLR)

Queue less than comparison.

### Syntax

```

template<typename Value,
         typename Container>
bool operator<(queue<Value, Container>% left,
                  queue<Value, Container>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true if, for the lowest position *i* for which  $!(right[i] < left[i])$  it is also true

that `left[i] < right[i]`. Otherwise, it returns `left->queue::size (STL/CLR) () < right->size()`. You use it to test whether *left* is ordered before *right* when the two queues are compared element by element.

## Example

```
// cliext_queue_operator_lt.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

## operator<= (queue) (STL/CLR)

Queue less than or equal comparison.

### Syntax

```
template<typename Value,
         typename Container>
bool operator<=(queue<Value, Container>% left,
                 queue<Value, Container>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two queues are compared element by element.

## Example

```
// cliext_queue_operator_le.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator== (queue) (STL/CLR)

Queue equal comparison.

## Syntax

```
template<typename Value,
         typename Container>
bool operator==(queue<Value, Container>% left,
                  queue<Value, Container>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position *i*, *left[i] == right[i]*. You use it to test whether *left* is ordered the same as *right* when the two queues are compared element by element.

## Example

```
// cliext_queue_operator_eq.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

## operator> (queue) (STL/CLR)

Queue greater than comparison.

## Syntax

```
template<typename Value,
         typename Container>
bool operator>(queue<Value, Container>% left,
                  queue<Value, Container>% right);
```

#### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

#### Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two queues are compared element by element.

#### Example

```
// cliext_queue_operator_gt.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
                           c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
                           c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

`operator>= (queue) (STL/CLR)`

Queue greater than or equal comparison.

## Syntax

```
template<typename Value,
         typename Container>
bool operator>=(queue<Value, Container>% left,
                  queue<Value, Container>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two queues are compared element by element.

### Example

```
// cliext_queue_operator_ge.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

# set (STL/CLR)

9/20/2022 • 37 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has bidirectional access. You use the container `set` to manage a sequence of elements as a (nearly) balanced ordered tree of nodes, each storing one element.

In the description below, `GValue` is the same as `GKey`, which in turn is the same as `Key` unless the latter is a ref type, in which case it is `Key^`.

## Syntax

```
template<typename Key>
ref class set
    : public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
System::Collections::Generic::IList<GValue>,
Microsoft::VisualC::StlClr::ITree<Gkey, GValue>
{ ..... };
```

## Parameters

### Key

The type of the key component of an element in the controlled sequence.

## Requirements

**Header:** <cliext/set>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">set::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.
<a href="#">set::const_reference (STL/CLR)</a>	The type of a constant reference to an element.
<a href="#">set::const_reverse_iterator (STL/CLR)</a>	The type of a constant reverse iterator for the controlled sequence.
<a href="#">set::difference_type (STL/CLR)</a>	The type of a (possibly signed) distance between two elements.
<a href="#">set::generic_container (STL/CLR)</a>	The type of the generic interface for the container.

TYPE DEFINITION	DESCRIPTION
<code>set::generic_iterator (STL/CLR)</code>	The type of an iterator for the generic interface for the container.
<code>set::generic_reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the generic interface for the container.
<code>set::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container.
<code>set::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>set::key_compare (STL/CLR)</code>	The ordering delegate for two keys.
<code>set::key_type (STL/CLR)</code>	The type of an ordering key.
<code>set::reference (STL/CLR)</code>	The type of a reference to an element.
<code>set::reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the controlled sequence.
<code>set::size_type (STL/CLR)</code>	The type of a (non-negative) distance between two elements.
<code>set::value_compare (STL/CLR)</code>	The ordering delegate for two element values.
<code>set::value_type (STL/CLR)</code>	The type of an element.
MEMBER FUNCTION	DESCRIPTION
<code>set::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>set::clear (STL/CLR)</code>	Removes all elements.
<code>set::count (STL/CLR)</code>	Counts elements matching a specified key.
<code>set::empty (STL/CLR)</code>	Tests whether no elements are present.
<code>set::end (STL/CLR)</code>	Designates the end of the controlled sequence.
<code>set::equal_range (STL/CLR)</code>	Finds range that matches a specified key.
<code>set::erase (STL/CLR)</code>	Removes elements at specified positions.
<code>set::find (STL/CLR)</code>	Finds an element that matches a specified key.
<code>set::insert (STL/CLR)</code>	Adds elements.
<code>set::key_comp (STL/CLR)</code>	Copies the ordering delegate for two keys.
<code>set::lower_bound (STL/CLR)</code>	Finds beginning of range that matches a specified key.
<code>set::make_value (STL/CLR)</code>	Constructs a value object.

MEMBER FUNCTION	DESCRIPTION
<code>set::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.
<code>set::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>set::set (STL/CLR)</code>	Constructs a container object.
<code>set::size (STL/CLR)</code>	Counts the number of elements.
<code>set::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>set::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>set::upper_bound (STL/CLR)</code>	Finds end of range that matches a specified key.
<code>set::value_comp (STL/CLR)</code>	Copies the ordering delegate for two element values.
OPERATOR	DESCRIPTION
<code>set::operator= (STL/CLR)</code>	Replaces the controlled sequence.
<code>operator!= (set) (STL/CLR)</code>	Determines if a <code>set</code> object is not equal to another <code>set</code> object.
<code>operator&lt; (set) (STL/CLR)</code>	Determines if a <code>set</code> object is less than another <code>set</code> object.
<code>operator&lt;= (set) (STL/CLR)</code>	Determines if a <code>set</code> object is less than or equal to another <code>set</code> object.
<code>operator== (set) (STL/CLR)</code>	Determines if a <code>set</code> object is equal to another <code>set</code> object.
<code>operator&gt; (set) (STL/CLR)</code>	Determines if a <code>set</code> object is greater than another <code>set</code> object.
<code>operator&gt;= (set) (STL/CLR)</code>	Determines if a <code>set</code> object is greater than or equal to another <code>set</code> object.

## Interfaces

INTERFACE	DESCRIPTION
<code>ICloneable</code>	Duplicate an object.
<code>IEnumerable</code>	Sequence through elements.
<code>ICollection</code>	Maintain group of elements.
<code>IEnumerable&lt;T&gt;</code>	Sequence through typed elements.

INTERFACE	DESCRIPTION
<code>ICollection&lt;T&gt;</code>	Maintain group of typed elements.
<code>ITree&lt;Key, Value&gt;</code>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls as individual nodes. It inserts elements into a (nearly) balanced tree that it keeps ordered by altering the links between nodes, never by copying the contents of one node to another. That means you can insert and remove elements freely without disturbing remaining elements.

The object orders the sequence it controls by calling a stored delegate object of type `set::key_compare (STL/CLR)`. You can specify the stored delegate object when you construct the set; if you specify no delegate object, the default is the comparison `operator<(key_type, key_type)`. You access this stored object by calling the member function `set::key_comp (STL/CLR)()`.

Such a delegate object must impose a strict weak ordering on keys of type `set::key_type (STL/CLR)`. That means, for any two keys `x` and `y`:

`key_comp()(x, y)` returns the same Boolean result on every call.

If `key_comp()(x, y)` is true, then `key_comp()(y, x)` must be false.

If `key_comp()(x, y)` is true, then `x` is said to be ordered before `y`.

If `!key_comp()(x, y) && !key_comp()(y, x)` is true, then `x` and `y` are said to have equivalent ordering.

For any element `x` that precedes `y` in the controlled sequence, `key_comp()(y, x)` is false. (For the default delegate object, keys never decrease in value.) Unlike template class `set`, an object of template class `set` does not require that keys for all elements are unique. (Two or more keys can have equivalent ordering.)

Each element serves as both a key and a value. The sequence is represented in a way that permits lookup, insertion, and removal of an arbitrary element with a number of operations proportional to the logarithm of the number of elements in the sequence (logarithmic time). Moreover, inserting an element invalidates no iterators, and removing an element invalidates only those iterators which point at the removed element.

A set supports bidirectional iterators, which means you can step to adjacent elements given an iterator that designates an element in the controlled sequence. A special head node corresponds to the iterator returned by `set::end (STL/CLR)()`. You can decrement this iterator to reach the last element in the controlled sequence, if present. You can increment a set iterator to reach the head node, and it will then compare equal to `end()`. But you cannot dereference the iterator returned by `end()`.

Note that you cannot refer to a set element directly given its numerical position -- that requires a random-access iterator.

A set iterator stores a handle to its associated set node, which in turn stores a handle to its associated container. You can use iterators only with their associated container objects. A set iterator remains valid so long as its associated set node is associated with some set. Moreover, a valid iterator is dereferencable -- you can use it to access or alter the element value it designates -- so long as it is not equal to `end()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does *not* destroy its elements.

# Members

## set::begin (STL/CLR)

Designates the beginning of the controlled sequence.

### Syntax

```
iterator begin();
```

### Remarks

The member function returns a bidirectional iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

### Example

```
// cliext_set_begin.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myset::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("**begin() = {0}", *++it);
    return (0);
}
```

```
a b c
*begin() = a
**begin() = b
```

## set::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

### Remarks

The member function effectively calls [set::erase \(STL/CLR\)](#) ( [set::begin \(STL/CLR\)](#) (), [set::end \(STL/CLR\)](#) () ).

You use it to ensure that the controlled sequence is empty.

### Example

```
// cliext_set_clear.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
size() = 0
a b
size() = 0
```

## set::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

### Syntax

```
typedef T2 const_iterator;
```

### Remarks

The type describes an object of unspecified type `T2` that can serve as a constant bidirectional iterator for the controlled sequence.

### Example

```

// cliext_set_const_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myset::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## set::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_set_const_reference.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myset::const_iterator cit = c1.begin();
    for ( ; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Myset::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

## set::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```
// cliext_set_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myset::const_reverse_iterator crit = c1.rbegin();
    for ( ; crit != c1.rend(); ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## set::count (STL/CLR)

Finds the number of elements matching a specified key.

### Syntax

```
size_type count(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function returns the number of elements in the controlled sequence that have equivalent ordering with *key*. You use it to determine the number of elements currently in the controlled sequence that match a specified key.

## Example

```
// cliext_set_count.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
a b c
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

## set::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a possibly negative element count.

## Example

```

// cliext_set_difference_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myset::difference_type diff = 0;
    for (Myset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Myset::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

## set::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

### Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [set::size \(STL/CLR\) \(\) == 0](#). You use it to test whether the set is empty.

### Example

```

// cliext_set_empty.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

## set::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a bidirectional iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the end of the controlled sequence; its status doesn't change if the length of the controlled sequence changes.

### Example

```

// cliext_set_end.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    Myset::iterator it = c1.end();
    --it;
    System::Console::WriteLine("--- --end() = {0}", *--it);
    System::Console::WriteLine("---end() = {0}", *++it);
    return (0);
}

```

```

a b c
*-- --end() = b
*--end() = c

```

## set::equal\_range (STL/CLR)

Finds range that matches a specified key.

### Syntax

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

### Parameters

#### *key*

Key value to search for.

### Remarks

The member function returns a pair of iterators `cliext::pair<iterator, iterator>( set::lower_bound (STL/CLR) (key), set::upper_bound (STL/CLR) (key))`. You use it to determine the range of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_set_equal_range.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
typedef Myset::pair_iter_pair Pairii;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("{0} ", *pair1.first);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
equal_range(L'x') empty = True
b

```

## set::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
size_type erase(key_type key)

```

### Parameters

*first*

Beginning of range to erase.

*key*

Key value to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*, and returns an

iterator that designates the first element remaining beyond the element removed, or `set::end (STL/CLR) ()` if no such element exists. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [`first`, `last`), and returns an iterator that designates the first element remaining beyond any elements removed, or `end()` if no such element exists.. You use it to remove zero or more contiguous elements.

The third member function removes any element of the controlled sequence whose key has equivalent ordering to `key`, and returns a count of the number of elements removed. You use it to remove and count all elements that match a specified key.

Each element erasure takes time proportional to the logarithm of the number of elements in the controlled sequence.

## Example

```
// cliext_set_erase.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.insert(L'd');
    c1.insert(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    Myset::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

## set::find (STL/CLR)

Finds an element that matches a specified key.

## Syntax

```
iterator find(key_type key);
```

## Parameters

### key

Key value to search for.

## Remarks

If at least one element in the controlled sequence has equivalent ordering with *key*, the member function returns an iterator designating one of those elements; otherwise it returns `set::end (STL/CLR)` (). You use it to locate an element currently in the controlled sequence that matches a specified key.

## Example

```
// cliext_set_find.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());
    System::Console::WriteLine("find {0} = {1}",
        L'b', *c1.find(L'b'));
    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
a b c
find A = False
find b = b
find C = False
```

## set::generic\_container (STL/CLR)

The type of the generic interface for the container.

## Syntax

```
typedef Microsoft::VisualC::StlClr::
    ITree<GKey, GValue>
generic_container;
```

## Remarks

The type describes the generic interface for this template container class.

## Example

```
// cliext_set_generic_container.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(L'e');
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
a b c d
a b c d e
```

## set::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

## Example

```

// cliext_set_generic_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myset::generic_iterator gcit = gc1->begin();
    Myset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

## set::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value>
generic_reverse_iterator;

```

### Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_set_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myset::generic_reverse_iterator gcit = gc1->rbegin();
    Myset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
c

```

## set::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```

typedef GValue generic_value;

```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

### Example

```

// cliext_set_generic_value.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myset::generic_iterator gcit = gc1->begin();
    Myset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

## set::insert (STL/CLR)

Adds elements.

### Syntax

```

cliext::pair<iterator, bool> insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);

```

### Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Key value to insert.

*where*

Where in container to insert (hint only).

## Remarks

Each of the member functions inserts a sequence specified by the remaining operands.

The first member function endeavors to insert an element with value *val*, and returns a pair of values `x`. If `x.second` is true, `x.first` designates the newly inserted element; otherwise `x.first` designates an element with equivalent ordering that already exists and no new element is inserted. You use it to insert a single element.

The second member function inserts an element with value *val*, using *where* as a hint (to improve performance), and returns an iterator that designates the newly inserted element. You use it to insert a single element which might be adjacent to an element you know.

The third member function inserts the sequence `[first, last]`. You use it to insert zero or more elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

Each element insertion takes time proportional to the logarithm of the number of elements in the controlled sequence. Insertion can occur in amortized constant time, however, given a hint that designates an element adjacent to the insertion point.

## Example

```

// cliext_set_insert.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
typedef Myset::pair_iter_bool Pairib;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Pairib pair1 = c1.insert(L'x');
    System::Console::WriteLine("insert(L'x') = [{0} {1}]",
        *pair1.first, pair1.second);

    pair1 = c1.insert(L'b');
    System::Console::WriteLine("insert(L'b') = [{0} {1}]",
        *pair1.first, pair1.second);

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value with hint
    System::Console::WriteLine("insert(begin(), L'y') = {0}",
        *c1.insert(c1.begin(), L'y'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    Myset c2;
    Myset::iterator it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    Myset c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(L'x') = [x True]
insert(L'b') = [b False]
a b c x
insert(begin(), L'y') = y
a b c x y
a b c x
a b c x y

```

## set::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

### Remarks

The type describes an object of unspecified type `T1` that can serve as a bidirectional iterator for the controlled sequence.

### Example

```
// cliext_set_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## set::key\_comp (STL/CLR)

Copies the ordering delegate for two keys.

### Syntax

```
key_compare^key_comp();
```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two keys.

### Example

```

// cliext_set_key_comp.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    Myset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

// test a different ordering rule
    Myset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## set::key\_compare (STL/CLR)

The ordering delegate for two keys.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its key arguments.

### Example

```

// cliext_set_key_compare.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    Myset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

// test a different ordering rule
    Myset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

## set::key\_type (STL/CLR)

The type of an ordering key.

### Syntax

```
typedef Key key_type;
```

### Remarks

The type is a synonym for the template parameter *Key*.

### Example

```

// cliext_set_key_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using key_type
    for (Myset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myset::key_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## set::lower\_bound (STL/CLR)

Finds beginning of range that matches a specified key.

### Syntax

```
iterator lower_bound(key_type key);
```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the first element  $x$  in the controlled sequence that has equivalent ordering to *key*. If no such element exists, it returns [set::end \(STL/CLR\)](#) (); otherwise it returns an iterator that designates  $x$ . You use it to locate the beginning of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_set_lower_bound.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    System::Console::WriteLine("*lower_bound(L'a') = {0}",
        *c1.lower_bound(L'a'));
    System::Console::WriteLine("*lower_bound(L'b') = {0}",
        *c1.lower_bound(L'b'));
    return (0);
}

```

```

a b c
lower_bound(L'x') == end() = True
*lower_bound(L'a') = a
*lower_bound(L'b') = b

```

## set::make\_value (STL/CLR)

Constructs a value object.

### Syntax

```
static value_type make_value(key_type key);
```

### Parameters

*key*

Key value to use.

### Remarks

The member function returns a `value_type` object whose key is *key*. You use it to compose an object suitable for use with several other member functions.

### Example

```

// cliext_set_make_value.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(Myset::make_value(L'a'));
    c1.insert(Myset::make_value(L'b'));
    c1.insert(Myset::make_value(L'c'));

    // display contents " a b c"
    for each (Myset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## set::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
set<Key>% operator=(set<Key>% right);
```

### Parameters

*right*

Container to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

### Example

```

// cliext_set_operator_as.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (Myset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2 = c1;
    // display contents " a b c"
    for each (Myset::value_type elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

## set::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

### Syntax

```

reverse_iterator rbegin();

```

### Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the **beginning** of the reverse sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_set_rbegin.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myset::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("**rbegin() = {0}", *++rit);
    return (0);
}

```

```

a b c
*rbegin() = c
**rbegin() = b

```

## set::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_set_reference.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        Myset::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## set::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

### Syntax

```
reverse_iterator rend();
```

### Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_set_rend.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myset::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine(" --- --rend() = {0}", *--rit);
    System::Console::WriteLine(" --- rend() = {0}", *++rit);
    return (0);
}

```

```

a b c
*-- --rend() = b
---rend() = a

```

## set::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

### Syntax

```
typedef T3 reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

### Example

```

// cliext_set_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myset::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}

```

c b a

## set::set (STL/CLR)

Constructs a container object.

### Syntax

```

set();
explicit set(key_compare^ pred);
set(set<Key>% right);
set(set<Key>^ right);
template<typename InIter>
    setset(InIter first, InIter last);
template<typename InIter>
    set(InIter first, InIter last,
        key_compare^ pred);
set(System::Collections::Generic::IEnumerable<GValue>^ right);
set(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);

```

### Parameters

*first*

Beginning of range to insert.

*last*

End of range to insert.

*pred*

Ordering predicate for the controlled sequence.

*right*

Object or range to insert.

### Remarks

The constructor:

`set();`

initializes the controlled sequence with no elements, with the default ordering predicate `key_compare()`. You use

it to specify an empty initial controlled sequence, with the default ordering predicate.

The constructor:

```
explicit set(key_compare^ pred);
```

initializes the controlled sequence with no elements, with the ordering predicate *pred*. You use it to specify an empty initial controlled sequence, with the specified ordering predicate.

The constructor:

```
set(set<Key>% right);
```

initializes the controlled sequence with the sequence [`right.begin()`, `right.end()`], with the default ordering predicate. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the set object *right*, with the default ordering predicate.

The constructor:

```
set(set<Key>^ right);
```

initializes the controlled sequence with the sequence [`right->begin()`, `right->end()`], with the default ordering predicate. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the set object *right*, with the default ordering predicate.

The constructor:

```
template<typename InIter> set(InIter first, InIter last);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the default ordering predicate. You use it to make the controlled sequence a copy of another sequence, with the default ordering predicate.

The constructor:

```
template<typename InIter> set(InIter first, InIter last, key_compare^ pred);
```

initializes the controlled sequence with the sequence [`first`, `last`], with the ordering predicate *pred*. You use it to make the controlled sequence a copy of another sequence, with the specified ordering predicate.

The constructor:

```
set(System::Collections::Generic::IEnumerable<Key>% right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the default ordering predicate. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the default ordering predicate.

The constructor:

```
set(System::Collections::Generic::IEnumerable<Key>% right, key_compare^ pred);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*, with the ordering predicate *pred*. You use it to make the controlled sequence a copy of another sequence described by an enumerator, with the specified ordering predicate.

## Example

```
// cliext_set_construct.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
```

```

    ```

// construct an empty container
Myset c1;
System::Console::WriteLine("size() = {0}", c1.size());

c1.insert(L'a');
c1.insert(L'b');
c1.insert(L'c');
for each (wchar_t elem in c1)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an ordering rule
Myset c2 = cliext::greater_equal<wchar_t>();
System::Console::WriteLine("size() = {0}", c2.size());

c2.insert(c1.begin(), c1.end());
for each (wchar_t elem in c2)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range
Myset c3(c1.begin(), c1.end());
for each (wchar_t elem in c3)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Myset c4(c1.begin(), c1.end(),
        cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c4)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration
Myset c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
for each (wchar_t elem in c5)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myset c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c6)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct from a generic container
Myset c7(c4);
for each (wchar_t elem in c7)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct by copying another container
Myset c8(%c3);
for each (wchar_t elem in c8)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
a b c
size() = 0
c b a
c b a
a b c
c b a
c b a
a b c
```

## set::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [set::empty \(STL/CLR\)](#).

### Example

```
// cliext_set_size.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}
```

```
a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2
```

## set::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```
// cliext_set_size_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myset::size_type diff = 0;
    for (Myset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
```

## set::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```
void swap(set<Key>% right);
```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_set_swap.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Myset c2;
    c2.insert(L'd');
    c2.insert(L'e');
    c2.insert(L'f');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
d e f
d e f
a b c

```

## set::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```

cli::array<value_type>^ to_array();

```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```

// cliext_set_to_array.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c d
a b c

```

## set::upper\_bound (STL/CLR)

Finds end of range that matches a specified key.

### Syntax

```

iterator upper_bound(key_type key);

```

### Parameters

*key*

Key value to search for.

### Remarks

The member function determines the last element *x* in the controlled sequence that has equivalent ordering to *key*. If no such element exists, or if *x* is the last element in the controlled sequence, it returns [set::end \(STL/CLR\)](#) (); otherwise it returns an iterator that designates the first element beyond *x*. You use it to locate the end of a sequence of elements currently in the controlled sequence that match a specified key.

### Example

```

// cliext_set_upper_bound.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    System::Console::WriteLine("*upper_bound(L'a') = {0}",
        *c1.upper_bound(L'a'));
    System::Console::WriteLine("*upper_bound(L'b') = {0}",
        *c1.upper_bound(L'b'));
    return (0);
}

```

```

a b c
upper_bound(L'x') == end() = True
*upper_bound(L'a') = b
*upper_bound(L'b') = c

```

## set::value\_comp (STL/CLR)

Copies the ordering delegate for two element values.

### Syntax

```

value_compare^ value_comp();

```

### Remarks

The member function returns the ordering delegate used to order the controlled sequence. You use it to compare two element values.

### Example

```

// cliext_set_value_comp.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    Myset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

```

## set::value\_compare (STL/CLR)

The ordering delegate for two element values.

### Syntax

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;

```

### Remarks

The type is a synonym for the delegate that determines the ordering of its value arguments.

### Example

```

// cliext_set_value_compare.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    Myset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```
compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

## set::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef generic_value value_type;
```

### Remarks

The type is a synonym for `generic_value`.

### Example

```
// cliext_set_value_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using value_type
    for (Myset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Myset::value_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## operator!= (set) (STL/CLR)

List not equal comparison.

### Syntax

```
template<typename Key>
bool operator!=(set<Key>% left,
                set<Key>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two sets are compared element by element.

## Example

```
// cliext_set_operator_ne.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}
```

```
a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True
```

## operator< (set) (STL/CLR)

List less than comparison.

## Syntax

```
template<typename Key>
bool operator<(set<Key>% left,
    set<Key>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns true if, for the lowest position `i` for which `!(right[i] < left[i])` it is also true that `left[i] < right[i]`. Otherwise, it returns `left->size() < right->size()`. You use it to test whether *left* is ordered before *right* when the two sets are compared element by element.

## Example

```
// cliext_set_operator_lt.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

## operator<= (set) (STL/CLR)

List less than or equal comparison.

## Syntax

```
template<typename Key>
bool operator<=(set<Key>% left,
    set<Key>% right);
```

#### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

#### Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two sets are compared element by element.

#### Example

```
// cliext_set_operator_le.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator== (set) (STL/CLR)

List equal comparison.

## Syntax

```
template<typename Key>
bool operator==(set<Key>% left,
                  set<Key>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position *i*, *left[i] == right[i]*. You use it to test whether *left* is ordered the same as *right* when the two sets are compared element by element.

### Example

```
// cliext_set_operator_eq.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
                             c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
                             c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

operator> (set) (STL/CLR)

List greater than comparison.

## Syntax

```
template<typename Key>
    bool operator>(set<Key>% left,
                      set<Key>% right);
```

## Parameters

/eft

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `right < left`. You use it to test whether `left` is ordered after `right` when the two sets are compared element by element.

## Example

```
// cliext_set_operator_gt.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}", c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}", c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

## operator>= (set) (STL/CLR)

List greater than or equal comparison.

### Syntax

```
template<typename Key>
bool operator>=(set<Key>% left,
                 set<Key>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two sets are compared element by element.

### Example

```

// cliext_set_operator_ge.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
        c1 >= c2);
    return (0);
}

```

```

a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False

```

# stack (STL/CLR)

9/20/2022 • 18 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has last-in first-out access. You use the container adapter `stack` to manage an underlying container as a push-down stack.

In the description below, `GValue` is the same as *Value* unless the latter is a ref type, in which case it is `Value^`. Similarly, `GContainer` is the same as *Container* unless the latter is a ref type, in which case it is `Container^`.

## Syntax

```
template<typename Value,
         typename Container>
ref class stack
    : public
        System::ICloneable,
        Microsoft::VisualC::StlClr::IStack<GValue, GContainer>
{ .... };
```

### Parameters

#### *Value*

The type of an element in the controlled sequence.

#### *Container*

The type of the underlying container.

## Requirements

**Header:** <cliext/stack>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">stack::const_reference (STL/CLR)</a>	The type of a constant reference to an element.
<a href="#">stack::container_type (STL/CLR)</a>	The type of the underlying container.
<a href="#">stack::difference_type (STL/CLR)</a>	The type of a signed distance between two elements.
<a href="#">stack::generic_container (STL/CLR)</a>	The type of the generic interface for the container adapter.
<a href="#">stack::generic_value (STL/CLR)</a>	The type of an element for the generic interface for the container adapter.
<a href="#">stack::reference (STL/CLR)</a>	The type of a reference to an element.
<a href="#">stack::size_type (STL/CLR)</a>	The type of a signed distance between two elements.

TYPE DEFINITION	DESCRIPTION
stack::value_type (STL/CLR)	The type of an element.
MEMBER FUNCTION	DESCRIPTION
stack::assign (STL/CLR)	Replaces all elements.
stack::empty (STL/CLR)	Tests whether no elements are present.
stack::get_container (STL/CLR)	Accesses the underlying container.
stack::pop (STL/CLR)	Removes the last element.
stack::push (STL/CLR)	Adds a new last element.
stack::size (STL/CLR)	Counts the number of elements.
stack::stack (STL/CLR)	Constructs a container object.
stack::top (STL/CLR)	Accesses the last element.
stack::to_array (STL/CLR)	Copies the controlled sequence to a new array.
PROPERTY	DESCRIPTION
stack::top_item (STL/CLR)	Accesses the last element.
OPERATOR	DESCRIPTION
stack::operator= (STL/CLR)	Replaces the controlled sequence.
operator!= (stack) (STL/CLR)	Determines if a stack object is not equal to another stack object.
operator< (stack) (STL/CLR)	Determines if a stack object is less than another stack object.
operator<= (stack) (STL/CLR)	Determines if a stack object is less than or equal to another stack object.
operator== (stack) (STL/CLR)	Determines if a stack object is equal to another stack object.
operator> (stack) (STL/CLR)	Determines if a stack object is greater than another stack object.
operator>= (stack) (STL/CLR)	Determines if a stack object is greater than or equal to another stack object.

## Interfaces

INTERFACE	DESCRIPTION
<a href="#">ICloneable</a>	Duplicate an object.
<a href="#">IStack&lt;Value, Container&gt;</a>	Maintain generic container adapter.

## Remarks

The object allocates and frees storage for the sequence it controls through an underlying container, of type *Container*, that stores *Value* elements and grows on demand. The object restricts access to pushing and popping just the last element, implementing a last-in first-out queue (also known as a LIFO queue, or stack).

## Members

### stack::assign (STL/CLR)

Replaces all elements.

#### Syntax

```
void assign(stack<Value, Container>% right);
```

#### Parameters

*right*

Container adapter to insert.

#### Remarks

The member function assigns `right.get_container()` to the underlying container. You use it to change the entire contents of the stack.

#### Example

```
// cliext_stack_assign.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign a repetition of values
    Mystack c2;
    c2.assign(c1);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c  
a b c
```

## stack::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```
// cliext_stack_const_reference.cpp  
// compile with: /clr  
#include <cliext/stack>  
  
typedef cliext::stack<wchar_t> Mystack;  
int main()  
{  
    Mystack c1;  
    c1.push(L'a');  
    c1.push(L'b');  
    c1.push(L'c');  
  
    // display reversed contents " c b a"  
    for ( ; !c1.empty(); c1.pop())  
        { // get a const reference to an element  
        Mystack::const_reference cref = c1.top();  
        System::Console::Write("{0} ", cref);  
        }  
    System::Console::WriteLine();  
    return (0);  
}
```

```
c b a
```

## stack::container\_type (STL/CLR)

The type of the underlying container.

### Syntax

```
typedef Container value_type;
```

### Remarks

The type is a synonym for the template parameter *Container*.

### Example

```
// cliext_stack_container_type.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c" using container_type
    Mystack::container_type wc1 = c1.get_container();
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

## stack::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a possibly negative element count.

### Example

```

// cliext_stack_difference_type.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute negative difference
    Mystack::difference_type diff = c1.size();
    c1.push(L'd');
    c1.push(L'e');
    diff -= c1.size();
    System::Console::WriteLine("pushing 2 = {0}", diff);

    // compute positive difference
    diff = c1.size();
    c1.pop();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("popping 3 = {0}", diff);
    return (0);
}

```

```

a b c
pushing 2 = -2
popping 3 = 3

```

## stack::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

### Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [stack::size \(STL/CLR\) \(\) == 0](#). You use it to test whether the stack is empty.

### Example

```

// cliext_stack_empty.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.pop();
    c1.pop();
    c1.pop();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

## stack::generic\_container (STL/CLR)

The type of the generic interface for the container adapter.

### Syntax

```

typedef Microsoft::VisualC::StlClr::IStack<Value>
    generic_container;

```

### Remarks

The type describes the generic interface for this template container adapter class.

### Example

```

// cliext_stack_generic_container.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get interface to container
    Mystack::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push(L'e');
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

## stack::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```

typedef GValue generic_value;

```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class. (`GValue` is either `value_type` or `value_type^` if `value_type` is a ref type.)

### Example

```

// cliext_stack_generic_value.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get interface to container
    Mystack::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display in reverse using generic_value
    for (; !gc1->empty(); gc1->pop())
    {
        Mystack::generic_value elem = gc1->top();

        System::Console::Write("{0} ", elem);
    }
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
c b a

```

## stack::get\_container (STL/CLR)

Accesses the underlying container.

### Syntax

```

container_type^ get_container();

```

### Remarks

The member function returns a handle for underlying container. You use it to bypass the restrictions imposed by the container wrapper.

### Example

```

// cliext_stack_get_container.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c" using container_type
    Mystack::container_type wc1 = c1.get_container();
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## stack::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
stack <Value, Container>% operator=(stack <Value, Container>% right);
```

### Parameters

*right*

Container adapter to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

### Example

```

// cliext_stack_operator_as.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2 = c1;
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

## stack::pop (STL/CLR)

Removes the last element.

### Syntax

```

void pop();

```

### Remarks

The member function removes the last element of the controlled sequence, which must be non-empty. You use it to shorten the stack by one element at the back.

### Example

```

// cliext_stack_pop.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop();
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b

```

## stack::push (STL/CLR)

Adds a new last element.

### Syntax

```

void push(value_type val);

```

### Remarks

The member function inserts an element with value `val` at the end of the controlled sequence. You use it to append another element to the stack.

### Example

```

// cliext_stack_push.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## stack::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

### Remarks

The type describes a reference to an element.

### Example

```

// cliext_stack_reference.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify top of stack and redisplay
    Mystack::reference ref = c1.top();
    ref = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c  
a b x
```

## stack::size (STL/CLR)

Counts the number of elements.

### Syntax

```
size_type size();
```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [stack::empty \(STL/CLR\) \(\)](#).

### Example

```
// cliext_stack_size.cpp  
// compile with: /clr  
#include <cliext/stack>  
  
typedef cliext::stack<wchar_t> Mystack;  
int main()  
{  
    Mystack c1;  
    c1.push(L'a');  
    c1.push(L'b');  
    c1.push(L'c');  
  
    // display initial contents " a b c"  
    for each (wchar_t elem in c1.get_container())  
        System::Console::Write("{0} ", elem);  
    System::Console::WriteLine();  
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());  
  
    // pop an item and reinspect  
    c1.pop();  
    System::Console::WriteLine("size() = {0} after popping", c1.size());  
  
    // add two elements and reinspect  
    c1.push(L'a');  
    c1.push(L'b');  
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());  
    return (0);  
}
```

```
a b c  
size() = 3 starting with 3  
size() = 2 after popping  
size() = 4 after adding 2
```

## stack::size\_type (STL/CLR)

The type of a signed distance between two element.

### Syntax

```
typedef int size_type;
```

## Remarks

The type describes a non-negative element count.

## Example

```
// cliext_stack_size_type.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mystack::size_type diff = c1.size();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("size difference = {0}", diff);
    return (0);
}
```

```
a b c
size difference = 2
```

## stack::stack (STL/CLR)

Constructs a container adapter object.

### Syntax

```
stack();
stack(stack<Value, Container>% right);
stack(stack<Value, Container>^ right);
explicit stack(container_type% wrapped);
```

### Parameters

*right*

Object to copy.

*wrapped*

Wrapped container to use.

## Remarks

The constructor:

```
stack();
```

creates an empty wrapped container. You use it to specify an empty initial controlled sequence.

The constructor:

```
stack(stack<Value, Container>% right);
```

creates a wrapped container that is a copy of `right.get_container()`. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the stack object `right`.

The constructor:

```
stack(stack<Value, Container>^ right);
```

creates a wrapped container that is a copy of `right->get_container()`. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the stack object `*right`.

The constructor:

```
explicit stack(container_type% wrapped);
```

uses the existing container `wrapped` as the wrapped container. You use it to construct a stack from an existing container.

## Example

```
// cliext_stack_construct.cpp
// compile with: /clr
#include <cliext/stack>
#include <cliext/vector>

typedef cliext::stack<wchar_t> Mystack;
typedef cliext::vector<wchar_t> Myvector;
typedef cliext::stack<wchar_t, Myvector> Mystack_vec;
int main()
{
    //
    // construct an empty container
    Mystack c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    //
    // construct from an underlying container
    Myvector v2(5, L'x');
    Mystack_vec c2(v2);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    //
    // construct by copying another container
    Mystack_vec c3(c2);
    for each (wchar_t elem in c3.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    //
    // construct by copying another container through handle
    Mystack_vec c4(%c2);
    for each (wchar_t elem in c4.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
size() = 0
x x x x x
x x x x x
x x x x x
```

## stack::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```
cli::array<Value>^ to_array();
```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```
// cliext_stack_to_array.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c d
a b c
```

## stack::top (STL/CLR)

Accesses the last element.

### Syntax

```
reference top();
```

## Remarks

The member function returns a reference to the last element of the controlled sequence, which must be non-empty. You use it to access the last element, when you know it exists.

## Example

```
// cliext_stack_top.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("top() = {0}", c1.top());

    // alter last item and reinspect
    c1.top() = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
top() = c
a b x
```

## stack::top\_item (STL/CLR)

Accesses the last element.

### Syntax

```
property value_type top_item;
```

## Remarks

The property accesses the last element of the controlled sequence, which must be non-empty. You use it to read or write the last element, when you know it exists.

## Example

```

// cliext_stack_top_item.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("top_item = {0}", c1.top_item);

    // alter last item and reinspect
    c1.top_item = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
top_item = c
a b x

```

## stack::value\_type (STL/CLR)

The type of an element.

### Syntax

```

typedef Value value_type;

```

### Remarks

The type is a synonym for the template parameter *Value*.

### Example

```

// cliext_stack_value_type.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " a b c" using value_type
    for ( ; !c1.empty(); c1.pop())
        { // store element in value_type object
        Mystack::value_type val = c1.top();

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

c b a

## operator!= (stack) (STL/CLR)

Stack not equal comparison.

### Syntax

```

template<typename Value,
         typename Container>
bool operator!=(stack<Value, Container>% left,
                  stack<Value, Container>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two stacks are compared element by element.

### Example

```

// cliext_stack_operator_ne.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

## operator< (stack) (STL/CLR)

Stack less than comparison.

### Syntax

```

template<typename Value,
         typename Container>
bool operator<(stack<Value, Container>% left,
                  stack<Value, Container>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true if, for the lowest position *i* for which  $!(right[i] < left[i])$  it is also true

that `left[i] < right[i]`. Otherwise, it returns `left->stack::size (STL/CLR) () < right->size()`. You use it to test whether *left* is ordered before *right* when the two stacks are compared element by element.

## Example

```
// cliext_stack_operator_lt.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

## operator<= (stack) (STL/CLR)

Stack less than or equal comparison.

### Syntax

```
template<typename Value,
         typename Container>
bool operator<=(stack<Value, Container>% left,
                 stack<Value, Container>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two stacks are compared element by element.

## Example

```
// cliext_stack_operator_le.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator== (stack) (STL/CLR)

Stack equal comparison.

## Syntax

```
template<typename Value,
         typename Container>
bool operator==(stack<Value, Container>% left,
                  stack<Value, Container>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position *i*, *left[i] == right[i]*. You use it to test whether *left* is ordered the same as *right* when the two stacks are compared element by element.

## Example

```
// cliext_stack_operator_eq.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

## operator> (stack) (STL/CLR)

Stack greater than comparison.

## Syntax

```
template<typename Value,
         typename Container>
bool operator>(stack<Value, Container>% left,
                  stack<Value, Container>% right);
```

#### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

#### Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two stacks are compared element by element.

#### Example

```
// cliext_stack_operator_gt.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
                           c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
                           c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

`operator>= (stack) (STL/CLR)`

Stack greater than or equal comparison.

## Syntax

```
template<typename Value,
         typename Container>
bool operator>=(stack<Value, Container>% left,
                  stack<Value, Container>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two stacks are compared element by element.

### Example

```
// cliext_stack_operator_ge.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

# utility (STL/CLR)

9/20/2022 • 8 minutes to read • [Edit Online](#)

Include the STL/CLR header `<cliext/utility>` to define the template class `pair` and several supporting template functions.

## Syntax

```
#include <utility>
```

## Requirements

**Header:** `<cliext/utility>`

**Namespace:** `cliext`

## Declarations

CLASS	DESCRIPTION
<a href="#">pair (STL/CLR)</a>	Wrap a pair of elements.
OPERATOR	DESCRIPTION
<a href="#">operator== (pair) (STL/CLR)</a>	Pair equal comparison.
<a href="#">operator!= (pair) (STL/CLR)</a>	Pair not equal comparison.
<a href="#">operator&lt; (pair) (STL/CLR)</a>	Pair less than comparison.
<a href="#">operator&lt;= (pair) (STL/CLR)</a>	Pair less than or equal comparison.
<a href="#">operator&gt; (pair) (STL/CLR)</a>	Pair greater than comparison.
<a href="#">operator&gt;= (pair) (STL/CLR)</a>	Pair greater than or equal comparison.
FUNCTION	DESCRIPTION
<a href="#">make_pair (STL/CLR)</a>	Make a pair from a pair of values.

## pair (STL/CLR)

The template class describes an object that wraps a pair of values.

### Syntax

```
template<typename Value1,
         typename Value2>
ref class pair;
```

#### Parameters

*Value1*

The type of first wrapped value.

*Value2*

The type of second wrapped value.

## Members

TYPE DEFINITION	DESCRIPTION
<a href="#">pair::first_type (STL/CLR)</a>	The type of the first wrapped value.
<a href="#">pair::second_type (STL/CLR)</a>	The type of the second wrapped value.
MEMBER OBJECT	DESCRIPTION
<a href="#">pair::first (STL/CLR)</a>	The first stored value.
<a href="#">pair::second (STL/CLR)</a>	The second stored value.
MEMBER FUNCTION	DESCRIPTION
<a href="#">pair::pair (STL/CLR)</a>	Constructs a pair object.
<a href="#">pair::swap (STL/CLR)</a>	Swaps the contents of two pairs.
OPERATOR	DESCRIPTION
<a href="#">pair::operator= (STL/CLR)</a>	Replaces the stored pair of values.

## Remarks

The object stores a pair of values. You use this template class to combine two values into a single object. Also, the object `cliext::pair` (described here) stores only managed types; to store a pair of unmanaged types use `std::pair`, declared in `<utility>`.

### [pair::first \(STL/CLR\)](#)

The first wrapped value.

#### Syntax

```
Value1 first;
```

#### Remarks

The object stores the first wrapped value.

## Example

```
// cliext_pair_first.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    cliext::pair<wchar_t, int>::first_type first_val = c1.first;
    cliext::pair<wchar_t, int>::second_type second_val = c1.second;
    System::Console::WriteLine("[{0}, {1}]", first_val, second_val);
    return (0);
}
```

```
[x, 3]
```

## pair::first\_type (STL/CLR)

The type of the first wrapped value.

### Syntax

```
typedef Value1 first_type;
```

### Remarks

The type is a synonym for the template parameter *Value1*.

## Example

```
// cliext_pair_first_type.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    cliext::pair<wchar_t, int>::first_type first_val = c1.first;
    cliext::pair<wchar_t, int>::second_type second_val = c1.second;
    System::Console::WriteLine("[{0}, {1}]", first_val, second_val);
    return (0);
}
```

```
[x, 3]
```

## pair::operator= (STL/CLR)

Replaces the stored pair of values.

### Syntax

```
pair<Value1, Value2>% operator=(pair<Value1, Value2>% right);
```

### Parameters

*right*

Pair to copy.

### Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the stored pair of values with a copy of the stored pair of values in *right*.

### Example

```
// cliext_pair_operator_as.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    // assign to a new pair
    cliext::pair<wchar_t, int> c2;
    c2 = c1;
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);
    return (0);
}
```

```
[x, 3]
[x, 3]
```

## pair::pair (STL/CLR)

Constructs a pair object.

### Syntax

```
pair();
pair(pair<Coll>% right);
pair(pair<Coll>^ right);
pair(Value1 val1, Value2 val2);
```

### Parameters

*right*

Pair to store.

*val1*

First value to store.

*val2*

Second value to store.

### Remarks

The constructor:

```
pair();
```

initializes the stored pair with default constructed values.

The constructor:

```
pair(pair<Value1, Value2>% right);
```

initializes the stored pair with `right.` `pair::first (STL/CLR)` and `right.` `pair::second (STL/CLR)`.

```
pair(pair<Value1, Value2>^ right);
```

initializes the stored pair with `right->` `pair::first (STL/CLR)` and `right>` `pair::second (STL/CLR)`.

The constructor:

```
pair(Value1 val1, Value2 val2);
```

initializes the stored pair with `val1` and `val2`.

## Example

```
// cliext_pair_construct.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
// construct an empty container
    cliext::pair<wchar_t, int> c1;
    System::Console::WriteLine("[{0}, {1}]",
        c1.first == L'\0' ? "\\\0" : "??", c1.second);

// construct with a pair of values
    cliext::pair<wchar_t, int> c2(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

// construct by copying another pair
    cliext::pair<wchar_t, int> c3(c2);
    System::Console::WriteLine("[{0}, {1}]", c3.first, c3.second);

// construct by copying a pair handle
    cliext::pair<wchar_t, int> c4(%c3);
    System::Console::WriteLine("[{0}, {1}]", c4.first, c4.second);

    return (0);
}
```

```
[\0, 0]
[x, 3]
[x, 3]
[x, 3]
```

## pair::second (STL/CLR)

The second wrapped value.

### Syntax

```
Value2 second;
```

### Remarks

The object stores the second wrapped value.

## Example

```
// cliext_pair_second.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    cliext::pair<wchar_t, int>::first_type first_val = c1.first;
    cliext::pair<wchar_t, int>::second_type second_val = c1.second;
    System::Console::WriteLine("[{0}, {1}]", first_val, second_val);
    return (0);
}
```

```
[x, 3]
```

## pair::second\_type (STL/CLR)

The type of the second wrapped value.

### Syntax

```
typedef Value2 second_type;
```

### Remarks

The type is a synonym for the template parameter *Value2*.

### Example

```
// cliext_pair_second_type.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    cliext::pair<wchar_t, int>::first_type first_val = c1.first;
    cliext::pair<wchar_t, int>::second_type second_val = c1.second;
    System::Console::WriteLine("[{0}, {1}]", first_val, second_val);
    return (0);
}
```

```
[x, 3]
```

## pair::swap (STL/CLR)

Swaps the contents of two pairs.

### Syntax

```
void swap(pair<Value1, Value2>% right);
```

### Parameters

*right*

Pair to swap contents with.

## Remarks

The member function swaps the stored pair of values between `*this` and *right*.

## Example

```
// cliext_pair_swap.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    {
        cliext::deque<wchar_t> d1;
        d1.push_back(L'a');
        d1.push_back(L'b');
        d1.push_back(L'c');
        Mycoll c1(%d1);

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct another container with repetition of values
        cliext::deque<wchar_t> d2(5, L'x');
        Mycoll c2(%d2);
        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // swap and redisplay
        c1.swap(c2);
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}
```

```
a b c
x x x x x
x x x x x
a b c
```

## make\_pair (STL/CLR)

Make a `pair` from a pair of values.

## Syntax

```
template<typename Value1,
         typename Value2>
pair<Value1, Value2> make_pair(Value1 first, Value2 second);
```

## Parameters

*Value1*

The type of the first wrapped value.

*Value2*

The type of the second wrapped value.

*first*

First value to wrap.

*second*

Second value to wrap.

## Remarks

The template function returns `pair<Value1, Value2>(first, second)`. You use it to construct a `pair<Value1, Value2>` object from a pair of values.

## Example

```
// cliext_make_pair.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    c1 = cliext::make_pair(L'y', 4);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    return (0);
}
```

```
[x, 3]
[y, 4]
```

## operator!= (pair) (STL/CLR)

Pair not equal comparison.

## Syntax

```
template<typename Value1,
         typename Value2>
bool operator!=(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);
```

## Parameters

*left*

Left pair to compare.

*right*

Right pair to compare.

## Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two pairs are compared element by element.

## Example

```
// cliext_pair_operator_ne.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] != [x 3] is {0}",
        c1 != c1);
    System::Console::WriteLine("[x 3] != [x 4] is {0}",
        c1 != c2);
    return (0);
}
```

```
[x, 3]
[x, 4]
[x 3] != [x 3] is False
[x 3] != [x 4] is True
```

## operator< (pair) (STL/CLR)

Pair less than comparison.

### Syntax

```
template<typename Value1,
         typename Value2>
bool operator<(pair<Value1, Value2>% left,
                 pair<Value1, Value2>% right);
```

### Parameters

*left*

Left pair to compare.

*right*

Right pair to compare.

### Remarks

The operator function returns `left.first < right.first || !(right.first < left.first && left.second < right.second)`. You use it to test whether *left* is ordered before *right* when the two pairs are compared element by element.

## Example

```

// cliext_pair_operator_lt.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] < [x 3] is {0}",
        c1 < c1);
    System::Console::WriteLine("[x 3] < [x 4] is {0}",
        c1 < c2);
    return (0);
}

```

```

[x, 3]
[x, 4]
[x 3] < [x 3] is False
[x 3] < [x 4] is True

```

## operator<= (pair) (STL/CLR)

Pair less than or equal comparison.

### Syntax

```

template<typename Value1,
         typename Value2>
bool operator<=(pair<Value1, Value2>% left,
                 pair<Value1, Value2>% right);

```

### Parameters

*left*

Left pair to compare.

*right*

Right pair to compare.

### Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two pairs are compared element by element.

### Example

```

// cliext_pair_operator_le.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] <= [x 3] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[x 4] <= [x 3] is {0}",
        c2 <= c1);
    return (0);
}

```

```

[x, 3]
[x, 4]
[x 3] <= [x 3] is True
[x 4] <= [x 3] is False

```

## operator== (pair) (STL/CLR)

Pair equal comparison.

### Syntax

```

template<typename Value1,
         typename Value2>
bool operator==(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);

```

### Parameters

*left*

Left pair to compare.

*right*

Right pair to compare.

### Remarks

The operator function returns `left.first == right.first && left.second == right.second`. You use it to test whether *left* is ordered the same as *right* when the two pairs are compared element by element.

### Example

```

// cliext_pair_operator_eq.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] == [x 3] is {0}",
        c1 == c1);
    System::Console::WriteLine("[x 3] == [x 4] is {0}",
        c1 == c2);
    return (0);
}

```

```

[x, 3]
[x, 4]
[x 3] == [x 3] is True
[x 3] == [x 4] is False

```

## operator> (pair) (STL/CLR)

Pair greater than comparison.

### Syntax

```

template<typename Value1,
         typename Value2>
bool operator>(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);

```

### Parameters

*left*

Left pair to compare.

*right*

Right pair to compare.

### Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two pairs are compared element by element.

### Example

```

// cliext_pair_operator_gt.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] > [x 3] is {0}",
        c1 > c1);
    System::Console::WriteLine("[x 4] > [x 3] is {0}",
        c2 > c1);
    return (0);
}

```

```

[x, 3]
[x, 4]
[x 3] > [x 3] is False
[x 4] > [x 3] is True

```

## operator>= (pair) (STL/CLR)

Pair greater than or equal comparison.

### Syntax

```

template<typename Value1,
         typename Value2>
bool operator>=(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);

```

### Parameters

*left*

Left pair to compare.

*right*

Right pair to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two pairs are compared element by element.

### Example

```
// cliext_pair_operator_ge.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] >= [x 3] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[x 3] >= [x 4] is {0}",
        c1 >= c2);
    return (0);
}
```

```
[x, 3]
[x, 4]
[x 3] >= [x 3] is True
[x 3] >= [x 4] is False
```

# vector (STL/CLR)

9/20/2022 • 37 minutes to read • [Edit Online](#)

The template class describes an object that controls a varying-length sequence of elements that has random access. You use the container `vector` to manage a sequence of elements as a contiguous block of storage. The block is implemented as an array that grows on demand.

In the description below, `GValue` is the same as `Value` unless the latter is a ref type, in which case it is `Value^`.

## Syntax

```
template<typename Value>
ref class vector
    : public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    Microsoft::VisualC::StlClr::IVector<GValue>
{ ..... };
```

## Parameters

### `Value`

The type of an element in the controlled sequence.

## Requirements

**Header:** <cliext/vector>

**Namespace:** cliext

## Declarations

TYPE DEFINITION	DESCRIPTION
<a href="#">vector::const_iterator (STL/CLR)</a>	The type of a constant iterator for the controlled sequence.
<a href="#">vector::const_reference (STL/CLR)</a>	The type of a constant reference to an element.
<a href="#">vector::const_reverse_iterator (STL/CLR)</a>	The type of a constant reverse iterator for the controlled sequence.
<a href="#">vector::difference_type (STL/CLR)</a>	The type of a signed distance between two elements.
<a href="#">vector::generic_container (STL/CLR)</a>	The type of the generic interface for the container.
<a href="#">vector::generic_iterator (STL/CLR)</a>	The type of an iterator for the generic interface for the container.

TYPE DEFINITION	DESCRIPTION
<code>vector::generic_reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the generic interface for the container.
<code>vector::generic_value (STL/CLR)</code>	The type of an element for the generic interface for the container.
<code>vector::iterator (STL/CLR)</code>	The type of an iterator for the controlled sequence.
<code>vector::reference (STL/CLR)</code>	The type of a reference to an element.
<code>vector::reverse_iterator (STL/CLR)</code>	The type of a reverse iterator for the controlled sequence.
<code>vector::size_type (STL/CLR)</code>	The type of a signed distance between two elements.
<code>vector::value_type (STL/CLR)</code>	The type of an element.

MEMBER FUNCTION	DESCRIPTION
<code>vector::assign (STL/CLR)</code>	Replaces all elements.
<code>vector::at (STL/CLR)</code>	Accesses an element at a specified position.
<code>vector::back (STL/CLR)</code>	Accesses the last element.
<code>vector::begin (STL/CLR)</code>	Designates the beginning of the controlled sequence.
<code>vector::capacity (STL/CLR)</code>	Reports the size of allocated storage for the container.
<code>vector::clear (STL/CLR)</code>	Removes all elements.
<code>vector::empty (STL/CLR)</code>	Tests whether no elements are present.
<code>vector::end (STL/CLR)</code>	Designates the end of the controlled sequence.
<code>vector::erase (STL/CLR)</code>	Removes elements at specified positions.
<code>vector::front (STL/CLR)</code>	Accesses the first element.
<code>vector::insert (STL/CLR)</code>	Adds elements at a specified position.
<code>vector::pop_back (STL/CLR)</code>	Removes the last element.
<code>vector::push_back (STL/CLR)</code>	Adds a new last element.
<code>vector::rbegin (STL/CLR)</code>	Designates the beginning of the reversed controlled sequence.
<code>vector::rend (STL/CLR)</code>	Designates the end of the reversed controlled sequence.
<code>vector::reserve (STL/CLR)</code>	Ensures a minimum growth capacity for the container.

MEMBER FUNCTION	DESCRIPTION
<code>vector::resize (STL/CLR)</code>	Changes the number of elements.
<code>vector::size (STL/CLR)</code>	Counts the number of elements.
<code>vector::swap (STL/CLR)</code>	Swaps the contents of two containers.
<code>vector::to_array (STL/CLR)</code>	Copies the controlled sequence to a new array.
<code>vector::vector (STL/CLR)</code>	Constructs a container object.

PROPERTY	DESCRIPTION
<code>vector::back_item (STL/CLR)</code>	Accesses the last element.
<code>vector::front_item (STL/CLR)</code>	Accesses the first element.

OPERATOR	DESCRIPTION
<code>vector::operator= (STL/CLR)</code>	Replaces the controlled sequence.
<code>vector::operator[](STL/CLR)</code>	Accesses an element at a specified position.
<code>operator!= (vector) (STL/CLR)</code>	Determines if a <code>vector</code> object is not equal to another <code>vector</code> object.
<code>operator&lt; (vector) (STL/CLR)</code>	Determines if a <code>vector</code> object is less than another <code>vector</code> object.
<code>operator&lt;= (vector) (STL/CLR)</code>	Determines if a <code>vector</code> object is less than or equal to another <code>vector</code> object.
<code>operator== (vector) (STL/CLR)</code>	Determines if a <code>vector</code> object is equal to another <code>vector</code> object.
<code>operator&gt; (vector) (STL/CLR)</code>	Determines if a <code>vector</code> object is greater than another <code>vector</code> object.
<code>operator&gt;= (vector) (STL/CLR)</code>	Determines if a <code>vector</code> object is greater than or equal to another <code>vector</code> object.

## Interfaces

INTERFACE	DESCRIPTION
<code>ICloneable</code>	Duplicate an object.
<code>IEnumerable</code>	Sequence through elements.
<code>ICollection</code>	Maintain group of elements.

INTERFACE	DESCRIPTION
<code>IEnumerable&lt;T&gt;</code>	Sequence through typed elements.
<code>ICollection&lt;T&gt;</code>	Maintain group of typed elements.
<code>IList&lt;T&gt;</code>	Maintain ordered group of typed elements.
<code>IVector&lt;Value&gt;</code>	Maintain generic container.

## Remarks

The object allocates and frees storage for the sequence it controls through a stored array of `Value` elements, which grows on demand. Growth occurs in such a way that the cost of appending a new element is amortized constant time. In other words, the cost of adding elements at the end does not increase, on average, as the length of the controlled sequence gets larger. Thus, a vector is a good candidate for the underlying container for template class [stack \(STL/CLR\)](#).

A `vector` supports random-access iterators, which means you can refer to an element directly given its numerical position, counting from zero for the first (front) element, to `size() - 1` for the last (back) element. It also means that a vector is a good candidate for the underlying container for template class [priority\\_queue \(STL/CLR\)](#).

A vector iterator stores a handle to its associated vector object, along with the bias of the element it designates. You can use iterators only with their associated container objects. The bias of a vector element is the same as its position.

Inserting or erasing elements can change the element value stored at a given position, so the value designated by an iterator can also change. (The container may have to copy elements up or down to create a hole before an insert or to fill a hole after an erase.) Nevertheless, a vector iterator remains valid so long as its bias is in the range `[0, size()]`. Moreover, a valid iterator remains dereferencable -- you can use it to access or alter the element value it designates -- so long as its bias is not equal to `size()`.

Erasing or removing an element calls the destructor for its stored value. Destroying the container erases all elements. Thus, a container whose element type is a ref class ensures that no elements outlive the container. Note, however, that a container of handles does not destroy its elements.

## Members

### `vector::assign (STL/CLR)`

Replaces all elements.

#### Syntax

```
void assign(size_type count, value_type val);
template<typename InIt>
    void assign(InIt first, InIt last);
void assign(System::Collections::Generic::IEnumerable<Value>^ right);
```

#### Parameters

`count`

Number of elements to insert.

`first`

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Value of the element to insert.

## Remarks

The first member function replaces the controlled sequence with a repetition of *count* elements of value *val*. You use it to fill the container with elements all having the same value.

If *InIt* is an integer type, the second member function behaves the same as

`assign((size_type)first, (value_type)last)`. Otherwise, it replaces the controlled sequence with the sequence [`first`, `last`). You use it to make the controlled sequence a copy another sequence.

The third member function replaces the controlled sequence with the sequence designated by the enumerator *right*. You use it to make the controlled sequence a copy of a sequence described by an enumerator.

## Example

```
// cliext_vector_assign.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // assign a repetition of values
    cliext::vector<wchar_t> c2;
    c2.assign(6, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an iterator range
    c2.assign(c1.begin(), c1.end() - 1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an enumeration
    c2.assign( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
x x x x x x
a b
a b c
```

## vector::at (STL/CLR)

Accesses an element at a specified position.

### Syntax

```
reference at(size_type pos);
```

### Parameters

*pos*

Position of element to access.

### Remarks

The member function returns a reference to the element of the controlled sequence at position *pos*. You use it to read or write an element whose position you know.

### Example

```
// cliext_vector_at.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using at
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1.at(i));
    System::Console::WriteLine();

    // change an entry and redisplay
    c1.at(1) = L'x';
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a x c
```

## vector::back (STL/CLR)

Accesses the last element.

### Syntax

```
reference back();
```

### Remarks

The member function returns a reference to the last element of the controlled sequence, which must be non-empty. You use it to access the last element, when you know it exists.

### Example

```

// cliext_vector_back.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back() = {0}", c1.back());

    // alter last item and reinspect
    c1.back() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
back() = c
a b x

```

## vector::back\_item (STL/CLR)

Accesses the last element.

### Syntax

```
property value_type back_item;
```

### Remarks

The property accesses the last element of the controlled sequence, which must be non-empty. You use it to read or write the last element, when you know it exists.

### Example

```

// cliext_vector_back_item.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back_item = {0}", c1.back_item);

    // alter last item and reinspect
    c1.back_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
back_item = c
a b x

```

## vector::begin (STL/CLR)

Designates the beginning of the controlled sequence.

### Syntax

```

iterator begin();

```

### Remarks

The member function returns a random-access iterator that designates the first element of the controlled sequence, or just beyond the end of an empty sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_vector_begin.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::vector<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*begin() = a
*++begin() = b
x y c

```

## vector::capacity (STL/CLR)

Reports the size of allocated storage for the container.

### Syntax

```
size_type capacity();
```

### Remarks

The member function returns the storage currently allocated to hold the controlled sequence, a value at least as large as `vector::size (STL/CLR)()`. You use it to determine how much the container can grow before it must reallocate storage for the controlled sequence.

### Example

```

// cliext_vector_capacity.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1.at(i));
    System::Console::WriteLine();

    // increase capacity
    cliext::vector<wchar_t>::size_type cap = c1.capacity();
    System::Console::WriteLine("capacity() = {0}, ok = {1}",
        cap, c1.size() <= cap);
    c1.reserve(cap + 5);
    System::Console::WriteLine("capacity() = {0}, ok = {1}",
        c1.capacity(), cap + 5 <= c1.capacity());
    return (0);
}

```

```

a b c
capacity() = 4, ok = True
capacity() = 9, ok = True

```

## vector::clear (STL/CLR)

Removes all elements.

### Syntax

```
void clear();
```

### Remarks

The member function effectively calls `vector::erase (STL/CLR) ( vector::begin (STL/CLR) (), vector::end (STL/CLR) ())`. You use it to ensure that the controlled sequence is empty.

### Example

```

// cliext_vector_clear.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

a b c
size() = 0
a b
size() = 0

```

## vector::const\_iterator (STL/CLR)

The type of a constant iterator for the controlled sequence.

### Syntax

```

typedef T2 const_iterator;

```

### Remarks

The type describes an object of unspecified type `T2` that can serve as a constant random-access iterator for the controlled sequence.

### Example

```

// cliext_vector_const_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::vector<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}

```

a b c

## vector::const\_reference (STL/CLR)

The type of a constant reference to an element.

### Syntax

```
typedef value_type% const_reference;
```

### Remarks

The type describes a constant reference to an element.

### Example

```

// cliext_vector_const_reference.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::vector<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        cliext::vector<wchar_t>::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## vector::const\_reverse\_iterator (STL/CLR)

The type of a constant reverse iterator for the controlled sequence..

### Syntax

```
typedef T4 const_reverse_iterator;
```

### Remarks

The type describes an object of unspecified type `T4` that can serve as a constant reverse iterator for the controlled sequence.

### Example

```
// cliext_vector_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::vector<wchar_t>::const_reverse_iterator crit = c1.rbegin();
    cliext::vector<wchar_t>::const_reverse_iterator crend = c1.rend();
    for (; crit != crend; ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

## vector::difference\_type (STL/CLR)

The types of a signed distance between two elements.

### Syntax

```
typedef int difference_type;
```

### Remarks

The type describes a signed element count.

### Example

```

// cliext_vector_difference_type.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::vector<wchar_t>::difference_type diff = 0;
    for (cliext::vector<wchar_t>::iterator it = c1.begin();
         it != c1.end(); ++it) ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (cliext::vector<wchar_t>::iterator it = c1.end();
         it != c1.begin(); --it) --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

## vector::empty (STL/CLR)

Tests whether no elements are present.

### Syntax

```
bool empty();
```

### Remarks

The member function returns true for an empty controlled sequence. It is equivalent to [vector::size \(STL/CLR\)](#)  
(). == 0. You use it to test whether the vector is empty.

### Example

```

// cliext_vector_empty.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

## vector::end (STL/CLR)

Designates the end of the controlled sequence.

### Syntax

```
iterator end();
```

### Remarks

The member function returns a random-access iterator that points just beyond the end of the controlled sequence. You use it to obtain an iterator that designates the **current** end of the controlled sequence, but its status can change if the length of the controlled sequence changes.

### Example

```

// cliext_vector_end.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    cliext::vector<wchar_t>::iterator it = c1.end();
    --it;
    System::Console::WriteLine("---- --end() = {0}", *--it);
    System::Console::WriteLine("----end() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*-- --end() = b
*--end() = c
a x y

```

## vector::erase (STL/CLR)

Removes elements at specified positions.

### Syntax

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);

```

### Parameters

*first*

Beginning of range to erase.

*last*

End of range to erase.

*where*

Element to erase.

### Remarks

The first member function removes the element of the controlled sequence pointed to by *where*. You use it to remove a single element.

The second member function removes the elements of the controlled sequence in the range [`first`, `last`]. You

use it to remove zero or more contiguous elements.

Both member functions return an iterator that designates the first element remaining beyond any elements removed, or `vector::end (STL/CLR)` if no such element exists.

When erasing elements, the number of element copies is linear in the number of elements between the end of the erasure and the nearer end of the sequence. (When erasing one or more elements at either end of the sequence, no element copies occur.)

## Example

```
// cliext_vector_erase.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.push_back(L'd');
    c1.push_back(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    cliext::vector<wchar_t>::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

## vector::front (STL/CLR)

Accesses the first element.

### Syntax

```
reference front();
```

### Remarks

The member function returns a reference to the first element of the controlled sequence, which must be non-empty. You use it to read or write the first element, when you know it exists.

### Example

```
// cliext_vector_front.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front() = {0}", c1.front());

    // alter first item and reinspect
    c1.front() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front() = a
x b c
```

## vector::front\_item (STL/CLR)

Accesses the first element.

### Syntax

```
property value_type front_item;
```

### Remarks

The property accesses the first element of the controlled sequence, which must be non-empty. You use it to read or write the first element, when you know it exists.

### Example

```

// cliext_vector_front_item.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front_item = {0}", c1.front_item);

    // alter first item and reinspect
    c1.front_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
front_item = a
x b c

```

## vector::generic\_container (STL/CLR)

The type of the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::
    IVector<generic_value>
generic_container;

```

### Remarks

The type describes the generic interface for this template container class.

### Example

```

// cliext_vector_generic_container.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::vector<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(gc1->end(), L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push_back(L'e');

    System::Collections::IEnumerator^ enum1 =
        gc1->GetEnumerator();
    while (enum1->MoveNext())
        System::Console::Write("{0} ", enum1->Current);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

## vector::generic\_iterator (STL/CLR)

The type of an iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerRandomAccessIterator<generic_value>
generic_iterator;

```

### Remarks

The type describes a generic iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_vector_generic_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::vector<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::vector<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::vector<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

## vector::generic\_reverse\_iterator (STL/CLR)

The type of a reverse iterator for use with the generic interface for the container.

### Syntax

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value> generic_reverse_iterator;

```

### Remarks

The type describes a generic reverse iterator that can be used with the generic interface for this template container class.

### Example

```

// cliext_vector_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::vector<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::vector<wchar_t>::generic_reverse_iterator gcit = gc1->rbegin();
    cliext::vector<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a c c

```

## vector::generic\_value (STL/CLR)

The type of an element for use with the generic interface for the container.

### Syntax

```
typedef GValue generic_value;
```

### Remarks

The type describes an object of type `GValue` that describes the stored element value for use with the generic interface for this template container class.

### Example

```

// cliext_vector_generic_value.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::vector<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::vector<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::vector<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

## vector::insert (STL/CLR)

Adds elements at a specified position.

### Syntax

```

iterator insert(iterator where, value_type val);
void insert(iterator where, size_type count, value_type val);
template<typename InIt>
    void insert(iterator where, InIt first, InIt last);
void insert(iterator where,
    System::Collections::Generic::IEnumerable<Value>^ right);

```

### Parameters

*count*

Number of elements to insert.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Enumeration to insert.

*val*

Value of the element to insert.

*where*

Where in container to insert before.

## Remarks

Each of the member functions inserts, before the element pointed to by *where* in the controlled sequence, a sequence specified by the remaining operands.

The first member function inserts an element with value *val* and returns an iterator that designates the newly inserted element. You use it to insert a single element before a place designated by an iterator.

The second member function inserts a repetition of *count* elements of value *val*. You use it to insert zero or more contiguous elements which are all copies of the same value.

If `Init` is an integer type, the third member function behaves the same as

`insert(where, (size_type)first, (value_type)last)`. Otherwise, it inserts the sequence `[ first, last ]`. You use it to insert zero or more contiguous elements copied from another sequence.

The fourth member function inserts the sequence designated by the *right*. You use it to insert a sequence described by an enumerator.

When inserting a single element, the number of element copies is linear in the number of elements between the insertion point and the nearer end of the sequence. (When inserting one or more elements at either end of the sequence, no element copies occur.) If `Init` is an input iterator, the third member function effectively performs a single insertion for each element in the sequence. Otherwise, when inserting `N` elements, the number of element copies is linear in `N` plus the number of elements between the insertion point and the nearer end of the sequence.

## Example

```

// cliext_vector_insert.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value using iterator
    cliext::vector<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("insert(begin()+1, L'x') = {0}",
        *c1.insert(++it, L'x'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a repetition of values
    cliext::vector<wchar_t> c2;
    c2.insert(c2.begin(), 2, L'y');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    it = c1.end();
    c2.insert(c2.end(), c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    c2.insert(c2.begin(), // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(begin()+1, L'x') = x
a x b c
y y
y y a x b
a x b c y a x b

```

## vector::iterator (STL/CLR)

The type of an iterator for the controlled sequence.

### Syntax

```
typedef T1 iterator;
```

## Remarks

The type describes an object of unspecified type `T1` that can serve as a random-access iterator for the controlled sequence.

## Example

```
// cliext_vector_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::vector<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();

    // alter first element and redisplay
    it = c1.begin();
    *it = L'x';
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x b c
```

## vector::operator= (STL/CLR)

Replaces the controlled sequence.

### Syntax

```
vector<Value>% operator=(vector<Value>% right);
```

### Parameters

*right*

Container to copy.

## Remarks

The member operator copies *right* to the object, then returns `*this`. You use it to replace the controlled sequence with a copy of the controlled sequence in *right*.

## Example

```

// cliext_vector_operator_as.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

## vector::operator(STL/CLR)

Accesses an element at a specified position.

### Syntax

```

reference operator[](size_type pos);

```

### Parameters

*pos*

Position of element to access.

### Remarks

The member operator returns a reference to the element at position *pos*. You use it to access an element whose position you know.

### Example

```

// cliext_vector_operator_sub.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using subscripting
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();

    // change an entry and redisplay
    c1[1] = L'x';
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a x c

```

## vector::pop\_back (STL/CLR)

Removes the last element.

### Syntax

```

void pop_back();

```

### Remarks

The member function removes the last element of the controlled sequence, which must be non-empty. You use it to shorten the vector by one element at the back.

### Example

```

// cliext_vector_pop_back.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_back();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b

```

## vector::push\_back (STL/CLR)

Adds a new last element.

### Syntax

```
void push_back(value_type val);
```

### Remarks

The member function inserts an element with value `val` at the end of the controlled sequence. You use it to append another element to the vector.

### Example

```

// cliext_vector_push_back.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

## vector::rbegin (STL/CLR)

Designates the beginning of the reversed controlled sequence.

### Syntax

```
reverse_iterator rbegin();
```

### Remarks

The member function returns a reverse iterator that designates the last element of the controlled sequence, or just beyond the beginning of an empty sequence. Hence, it designates the **beginning** of the reverse sequence. You use it to obtain an iterator that designates the **current** beginning of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

### Example

```
// cliext_vector_rbegin.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    cliext::vector<wchar_t>::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("**rbegin() = {0}", *rit);
    System::Console::WriteLine("++rbegin() = {0}", ++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
*rbegin() = c
++rbegin() = b
a y x
```

## vector::reference (STL/CLR)

The type of a reference to an element.

### Syntax

```
typedef value_type% reference;
```

## Remarks

The type describes a reference to an element.

## Example

```
// cliext_vector_reference.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::vector<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        cliext::vector<wchar_t>::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();

    // modify contents " a b c"
    for (it = c1.begin(); it != c1.end(); ++it)
        { // get a reference to an element
        cliext::vector<wchar_t>::reference ref = *it;

        ref += (wchar_t)(L'A' - L'a');
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
A B C
```

## vector::rend (STL/CLR)

Designates the end of the reversed controlled sequence.

## Syntax

```
reverse_iterator rend();
```

## Remarks

The member function returns a reverse iterator that points just beyond the beginning of the controlled sequence. Hence, it designates the `end` of the reverse sequence. You use it to obtain an iterator that designates the `current` end of the controlled sequence seen in reverse order, but its status can change if the length of the controlled sequence changes.

## Example

```

// cliext_vector_rend.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::vector<wchar_t>::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("---- --rend() = {0}", *--rit);
    System::Console::WriteLine("----rend() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*-- --rend() = b
*--rend() = a
y x c

```

## vector::reserve (STL/CLR)

Ensures a minimum growth capacity for the container.

### Syntax

```
void reserve(size_type count);
```

### Parameters

*count*

New minimum capacity of the container.

### Remarks

The member function ensures that `capacity()` henceforth returns at least *count*. You use it to ensure that the container need not reallocate storage for the controlled sequence until it has grown to the specified size.

### Example

```

// cliext_vector_reserve.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1.at(i));
    System::Console::WriteLine();

    // increase capacity
    cliext::vector<wchar_t>::size_type cap = c1.capacity();
    System::Console::WriteLine("capacity() = {0}, ok = {1}",
        cap, c1.size() <= cap);
    c1.reserve(cap + 5);
    System::Console::WriteLine("capacity() = {0}, ok = {1}",
        c1.capacity(), cap + 5 <= c1.capacity());
    return (0);
}

```

```

a b c
capacity() = 4, ok = True
capacity() = 9, ok = True

```

## vector::resize (STL/CLR)

Changes the number of elements.

### Syntax

```

void resize(size_type new_size);
void resize(size_type new_size, value_type val);

```

### Parameters

*new\_size*

New size of the controlled sequence.

*val*

Value of the padding element.

### Remarks

The member functions both ensure that `vector::size (STL/CLR)()` henceforth returns *new\_size*. If it must make the controlled sequence longer, the first member function appends elements with value `value_type()`, while the second member function appends elements with value *val*. To make the controlled sequence shorter, both member functions effectively erase the last element `vector::size (STL/CLR)() - new_size` times. You use it to ensure that the controlled sequence has size *new\_size*, by either trimming or padding the current controlled sequence.

### Example

```

// cliext_vector_resize.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
// construct an empty container and pad with default values
    cliext::vector<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());
    c1.resize(4);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

// resize to empty
    c1.resize(0);
    System::Console::WriteLine("size() = {0}", c1.size());

// resize and pad
    c1.resize(5, L'x');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

size() = 0
0 0 0 0
size() = 0
x x x x x

```

## vector::reverse\_iterator (STL/CLR)

The type of a reverse iterator for the controlled sequence.

### Syntax

```

typedef T3 reverse_iterator;

```

### Remarks

The type describes an object of unspecified type `T3` that can serve as a reverse iterator for the controlled sequence.

### Example

```

// cliext_vector_reverse_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::vector<wchar_t>::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();

    // alter first element and redisplay
    rit = c1.rbegin();
    *rit = L'x';
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}

```

```

c b a
x b a

```

## vector::size (STL/CLR)

Counts the number of elements.

### Syntax

```

size_type size();

```

### Remarks

The member function returns the length of the controlled sequence. You use it to determine the number of elements currently in the controlled sequence. If all you care about is whether the sequence has nonzero size, see [vector::empty \(STL/CLR\)](#).

### Example

```

// cliext_vector_size.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

## vector::size\_type (STL/CLR)

The type of a signed distance between two elements.

### Syntax

```
typedef int size_type;
```

### Remarks

The type describes a non-negative element count.

### Example

```

// cliext_vector_size_type.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::vector<wchar_t>::size_type diff = c1.end() - c1.begin();
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3

```

## vector::swap (STL/CLR)

Swaps the contents of two containers.

### Syntax

```
void swap(vector<Value>% right);
```

### Parameters

*right*

Container to swap contents with.

### Remarks

The member function swaps the controlled sequences between `*this` and *right*. It does so in constant time and it throws no exceptions. You use it as a quick way to exchange the contents of two containers.

### Example

```

// cliext_vector_swap.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::vector<wchar_t> c2(5, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
x x x x x
x x x x x
a b c

```

## vector::to\_array (STL/CLR)

Copies the controlled sequence to a new array.

### Syntax

```
cli::array<Value>^ to_array();
```

### Remarks

The member function returns an array containing the controlled sequence. You use it to obtain a copy of the controlled sequence in array form.

### Example

```
// cliext_vector_to_array.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push_back(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c d
a b c
```

## vector::value\_type (STL/CLR)

The type of an element.

### Syntax

```
typedef Value value_type;
```

### Remarks

The type is a synonym for the template parameter *Value*.

### Example

```

// cliext_vector_value_type.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using value_type
    for (cliext::vector<wchar_t>::iterator it = c1.begin();
        it != c1.end(); ++it)
    {   // store element in value_type object
        cliext::vector<wchar_t>::value_type val = *it;

        System::Console::Write("{0} ", val);
    }
    System::Console::WriteLine();
    return (0);
}

```

a b c

## vector::vector (STL/CLR)

Constructs a container object.

### Syntax

```

vector();
vector(vector<Value>% right);
vector(vector<Value>^ right);
explicit vector(size_type count);
vector(size_type count, value_type val);
template<typename InIt>
    vector(InIt first, InIt last);
vector(System::Collections::Generic::IEnumerable<Value>^ right);

```

### Parameters

*count*

Number of elements to insert.

*first*

Beginning of range to insert.

*last*

End of range to insert.

*right*

Object or range to insert.

*val*

Value of the element to insert.

### Remarks

The constructor:

```
vector();
```

initializes the controlled sequence with no elements. You use it to specify an empty initial controlled sequence.

The constructor:

```
vector(vector<Value>% right);
```

initializes the controlled sequence with the sequence [ `right.begin()` , `right.end()` ]. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the vector object *right*.

The constructor:

```
vector(vector<Value>^ right);
```

initializes the controlled sequence with the sequence [ `right->begin()` , `right->end()` ]. You use it to specify an initial controlled sequence that is a copy of the sequence controlled by the vector object whose handle is *right*.

The constructor:

```
explicit vector(size_type count);
```

initializes the controlled sequence with *count* elements each with value `value_type()`. You use it to fill the container with elements all having the default value.

The constructor:

```
vector(size_type count, value_type val);
```

initializes the controlled sequence with *count* elements each with value *val*. You use it to fill the container with elements all having the same value.

The constructor:

```
template<typename InIt>
```

```
vector(InIt first, InIt last);
```

initializes the controlled sequence with the sequence [ `first` , `last` ]. You use it to make the controlled sequence a copy of another sequence.

The constructor:

```
vector(System::Collections::Generic::IEnumerable<Value>^ right);
```

initializes the controlled sequence with the sequence designated by the enumerator *right*. You use it to make the controlled sequence a copy of another sequence described by an enumerator.

## Example

```

// cliext_vector_construct.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
// construct an empty container
    cliext::vector<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());

// construct with a repetition of default values
    cliext::vector<wchar_t> c2(3);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

// construct with a repetition of values
    cliext::vector<wchar_t> c3(6, L'x');
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range
    cliext::vector<wchar_t>::iterator it = c3.end();
    cliext::vector<wchar_t> c4(c3.begin(), --it);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an enumeration
    cliext::vector<wchar_t> c5( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
    for each (wchar_t elem in c5)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying another container
    cliext::vector<wchar_t> c7(c3);
    for each (wchar_t elem in c7)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying a container handle
    cliext::vector<wchar_t> c8(%c3);
    for each (wchar_t elem in c8)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}

```

```

size() = 0
0 0 0
x x x x x x
x x x x x
x x x x x x
x x x x x x
x x x x x x

```

## operator!= (vector) (STL/CLR)

Vector not equal comparison.

### Syntax

```
template<typename Value>
bool operator!=(vector<Value>% left,
                 vector<Value>% right);
```

#### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

#### Remarks

The operator function returns `!(left == right)`. You use it to test whether *left* is not ordered the same as *right* when the two vectors are compared element by element.

#### Example

```
// cliext_vector_operator_ne.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
                           c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
                           c1 != c2);
    return (0);
}
```

```
a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True
```

## operator< (vector) (STL/CLR)

Vector less than comparison.

## Syntax

```
template<typename Value>
bool operator<(vector<Value>% left,
                 vector<Value>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns true if, for the lowest position *i* for which  $!(right[i] < left[i])$  it is also true that  $left[i] < right[i]$ . Otherwise, it returns  $left->size() < right->size()$ . You use it to test whether *left* is ordered before *right* when the two vectors are compared element by element.

### Example

```
// cliext_vector_operator_lt.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
                           c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
                           c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

Vector less than or equal comparison.

## Syntax

```
template<typename Value>
bool operator<=(vector<Value>% left,
                 vector<Value>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(right < left)`. You use it to test whether *left* is not ordered after *right* when the two vectors are compared element by element.

### Example

```
// cliext_vector_operator_le.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
                             c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
                             c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

## operator== (vector) (STL/CLR)

## Vector equal comparison.

## Syntax

```
template<typename Value>
    bool operator==(vector<Value>% left,
                      vector<Value>% right);
```

## Parameters

*left*

Left container to compare.

*right*

Right container to compare.

## Remarks

The operator function returns true only if the sequences controlled by *left* and *right* have the same length and, for each position `i`, `left[i] == right[i]`. You use it to test whether *left* is ordered the same as *right* when the two vectors are compared element by element.

## Example

```
// cliext_vector_operator_eq.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}", 
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}", 
        c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

## operator> (vector) (STL/CLR)

Vector greater than comparison.

### Syntax

```
template<typename Value>
bool operator>(vector<Value>% left,
                 vector<Value>% right);
```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `right < left`. You use it to test whether *left* is ordered after *right* when the two vectors are compared element by element.

### Example

```

// cliext_vector_operator_gt.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}

```

```

a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True

```

## operator>= (vector) (STL/CLR)

Vector greater than or equal comparison.

### Syntax

```

template<typename Value>
bool operator>=(vector<Value>% left,
                 vector<Value>% right);

```

### Parameters

*left*

Left container to compare.

*right*

Right container to compare.

### Remarks

The operator function returns `!(left < right)`. You use it to test whether *left* is not ordered before *right* when the two vectors are compared element by element.

## Example

```
// cliext_vector_operator_ge.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
        c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

# C++ Support Library

9/20/2022 • 2 minutes to read • [Edit Online](#)

The C++ Support Library provides classes that support managed programming in C++.

## In This Section

[Overview of Marshaling in C++](#)

[Resource Management Classes](#)

[Synchronization \(lock Class\)](#)

[Calling Functions in a Specific Application Domain](#)

[com::ptr](#)

# Overview of Marshaling in C++/CLI

9/20/2022 • 2 minutes to read • [Edit Online](#)

In mixed mode, you sometimes must marshal your data between native and managed types. The *marshaling library* helps you marshal and convert data in a simple way. The marshaling library consists of a set of functions and a `marshal_context` class that perform marshaling for common types. The library is defined in these headers in the `include/msclr` directory for your Visual Studio edition:

HEADER	DESCRIPTION
marshal.h	<code>marshal_context</code> class and context-free marshaling functions
marshal_atl.h	Functions for marshaling ATL types
marshal_cppstd.h	Functions for marshaling standard C++ types
marshal_windows.h	Functions for marshaling Windows types

The default path for `msclr` folder is something like this depending on which edition you have and the build number:

```
C:\\Program Files (x86)\\Microsoft Visual  
Studio\\Preview\\Enterprise\\VC\\Tools\\MSVC\\14.15.26528\\include\\msclr
```

You can use the marshaling library with or without a [marshal\\_context Class](#). Some conversions require a context. Other conversions can be implemented using the [marshal\\_as](#) function. The following table lists the current conversions supported, whether they require a context, and what marshal file you have to include:

FROM TYPE	TO TYPE	MARSHAL METHOD	INCLUDE FILE
System::String^	const char *	marshal_context	marshal.h
const char *	System::String^	marshal_as	marshal.h
char *	System::String^	marshal_as	marshal.h
System::String^	const wchar_t*	marshal_context	marshal.h
const wchar_t *	System::String^	marshal_as	marshal.h
wchar_t *	System::String^	marshal_as	marshal.h
System::IntPtr	HANDLE	marshal_as	marshal_windows.h
HANDLE	System::IntPtr	marshal_as	marshal_windows.h
System::String^	BSTR	marshal_context	marshal_windows.h

FROM TYPE	TO TYPE	MARSHAL METHOD	INCLUDE FILE
BSTR	System::String^	marshal_as	marshal.h
System::String^	bstr_t	marshal_as	marshal_windows.h
bstr_t	System::String^	marshal_as	marshal_windows.h
System::String^	std::string	marshal_as	marshal_cppstd.h
std::string	System::String^	marshal_as	marshal_cppstd.h
System::String^	std::wstring	marshal_as	marshal_cppstd.h
std::wstring	System::String^	marshal_as	marshal_cppstd.h
System::String^	CStringT<char>	marshal_as	marshal_atl.h
CStringT<char>	System::String^	marshal_as	marshal_atl.h
System::String^	CStringT<wchar_t>	marshal_as	marshal_atl.h
CStringT<wchar_t>	System::String^	marshal_as	marshal_atl.h
System::String^	CComBSTR	marshal_as	marshal_atl.h
CComBSTR	System::String^	marshal_as	marshal_atl.h

Marshaling requires a context only when you marshal from managed to native data types and the native type you are converting to does not have a destructor for automatic clean up. The marshaling context destroys the allocated native data type in its destructor. Therefore, conversions that require a context will be valid only until the context is deleted. To save any marshaled values, you must copy the values to your own variables.

#### NOTE

If you have embedded `NULL`s in your string, the result of marshaling the string is not guaranteed. The embedded `NULL`s can cause the string to be truncated or they might be preserved.

This example shows how to include the msclr directory in an include header declaration:

```
#include "msclr\marshal_cppstd.h"
```

The marshaling library is extensible so that you can add your own marshaling types. For more information about extending the marshaling library, see [How to: Extend the Marshaling Library](#).

## See also

[C++ Support Library](#)

[How to: Extend the Marshaling Library](#)

# marshal\_as

9/20/2022 • 2 minutes to read • [Edit Online](#)

This method converts data between native and managed environments.

## Syntax

```
To_Type marshal_as<To_Type>(
    From_Type input
);
```

### Parameters

*input*

[in] The value that you want to marshal to a `To_Type` variable.

## Return Value

A variable of type `To_Type` that is the converted value of `input`.

## Remarks

This method is a simplified way to convert data between native and managed types. To determine what data types are supported, see [Overview of Marshaling in C++](#). Some data conversions require a context. You can convert those data types by using the [marshal\\_context Class](#).

If you try to marshal a pair of data types that are not supported, `marshal_as` will generate an error [C4996](#) at compile time. Read the message supplied with this error for more information. The `c4996` error can be generated for more than just deprecated functions. One example of this is trying to marshal a pair of data types that are not supported.

The marshaling library consists of several header files. Any conversion requires only one file, but you can include additional files if you need to for other conversions. To see which conversions are associated with which files, look in the table in [Marshaling Overview](#). Regardless of what conversion you want to do, the namespace requirement is always in effect.

Throws `System::ArgumentNullException(_EXCEPTION_NULLPTR)` if the `input` parameter is null.

## Example

This example marshals from a `const char*` to a `System::String` variable type.

```
// marshal_as_test.cpp
// compile with: /clr
#include <stdlib.h>
#include <string.h>
#include <msclr\marshal.h>

using namespace System;
using namespace msclr::interop;

int main() {
    const char* message = "Test String to Marshal";
    String^ result;
    result = marshal_as<String^>( message );
    return 0;
}
```

## Requirements

**Header file:** <msclr\marshal.h>, <msclr\marshal\_windows.h>, <msclr\marshal\_cppstd.h>, or <msclr\marshal\_atl.h>

**Namespace:** msclr::interop

## See also

[Overview of Marshaling in C++](#)

[marshal\\_context Class](#)

# marshal\_context Class

9/20/2022 • 2 minutes to read • [Edit Online](#)

This class converts data between native and managed environments.

## Syntax

```
class marshal_context
```

## Remarks

Use the `marshal_context` class for data conversions that require a context. For more information about which conversions require a context and which marshaling file has to be included, see [Overview of marshaling in C++](#). The result of marshaling when you use a context is valid only until the `marshal_context` object is destroyed. To preserve your result, you must copy the data.

The same `marshal_context` can be used for numerous data conversions. Reusing the context in this manner won't affect the results from previous marshaling calls.

## Members

### Public constructors

NAME	DESCRIPTION
<code>marshal_context::marshal_context</code>	Constructs a <code>marshal_context</code> object to use for data conversion between managed and native data types.
<code>marshal_context::~marshal_context</code>	Destroys a <code>marshal_context</code> object.

### Public methods

NAME	DESCRIPTION
<code>marshal_context::marshal_as</code>	Performs the marshaling on a specific data object to convert it between a managed and a native data type.

## Requirements

**Header file:** <msclr\marshal.h>, <msclr\marshal\_windows.h>, <msclr\marshal\_cppstd.h>, or <msclr\marshal\_atl.h>

**Namespace:** msclr::interop

### marshal\_context::marshal\_context

Constructs a `marshal_context` object to use for data conversion between managed and native data types.

```
marshal_context();
```

## Remarks

Some data conversions require a marshal context. For more information about which translations require a context and which marshaling file you must include in your application, see [Overview of marshaling in C++](#).

## Example

See the example for [marshal\\_context::marshal\\_as](#).

## marshal\_context::~marshal\_context

Destroys a `marshal_context` object.

```
~marshal_context();
```

## Remarks

Some data conversions require a marshal context. See [Overview of marshaling in C++](#) for more information about which translations require a context and which marshaling file has to be included in your application.

Deleting a `marshal_context` object will invalidate the data converted by that context. If you want to preserve your data after a `marshal_context` object is destroyed, you must manually copy the data to a variable that will persist.

## marshal\_context::marshal\_as

Performs the marshaling on a specific data object to convert it between a managed and a native data type.

```
To_Type marshal_as<To_Type>(
    From_Type input
);
```

## Parameters

### *input*

[in] The value that you want to marshal to a `To_Type` variable.

## Return value

A variable of type `To_Type` that's the converted value of `input`.

## Remarks

This function performs the marshaling on a specific data object. Use this function only with the conversions indicated by the table in [Overview of marshaling in C++](#).

If you try to marshal a pair of data types that aren't supported, `marshal_as` will generate an error [C4996](#) at compile time. Read the message supplied with this error for more information. The `c4996` error can be generated for more than just deprecated functions. Two conditions that generate this error are trying to marshal a pair of data types that aren't supported and trying to use `marshal_as` for a conversion that requires a context.

The marshaling library consists of several header files. Any conversion requires only one file, but you can include additional files if you need to for other conversions. The table in [Marshaling Overview in C++](#) indicates which marshaling file should be included for each conversion.

## Example

This example creates a context for marshaling from a `System::String` to a `const char *` variable type. The converted data won't be valid after the line that deletes the context.

```
// marshal_context_test.cpp
// compile with: /clr
#include <stdlib.h>
#include <string.h>
#include <msclr\marshal.h>

using namespace System;
using namespace msclr::interop;

int main() {
    marshal_context^ context = gcnew marshal_context();
    String^ message = gcnew String("Test String to Marshal");
    const char* result;
    result = context->marshal_as<const char*>( message );
    delete context;
    return 0;
}
```

# msclr namespace

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `msclr` namespace contains all the classes of the C++ Support Library. For more information on those classes, see [C++ Support Library](#).

## See also

[C++ Support Library](#)

# Resource Management Classes

9/20/2022 • 2 minutes to read • [Edit Online](#)

These classes provide automatic management of managed classes.

## In This Section

### [auto\\_gcroot](#)

Embeds a virtual handle in a native type.

### [auto\\_handle](#)

Embeds a virtual handle in a managed type.

## See also

### [C++ Support Library](#)

# auto\_gcroot

9/20/2022 • 2 minutes to read • [Edit Online](#)

Defines the `auto_gcroot` class and `swap` function.

## Syntax

```
#include <msclr\auto_gcroot.h>
```

## Remarks

In this header file:

[auto\\_gcroot Class](#)

[swap Function \(auto\\_gcroot\)](#)

## See also

[C++ Support Library](#)

# auto\_gcroot Class

9/20/2022 • 8 minutes to read • [Edit Online](#)

Automatic resource management (like [auto\\_ptr Class](#)) which can be used to embed a virtual handle into a native type.

## Syntax

```
template<typename _element_type>
class auto_gcroot;
```

### Parameters

*\_element\_type*

The managed type to be embedded.

## Members

### Public constructors

NAME	DESCRIPTION
<a href="#">auto_gcroot::auto_gcroot</a>	The <code>auto_gcroot</code> constructor.
<a href="#">auto_gcroot::~auto_gcroot</a>	The <code>auto_gcroot</code> destructor.

### Public methods

NAME	DESCRIPTION
<a href="#">auto_gcroot::attach</a>	Attach <code>auto_gcroot</code> to an object.
<a href="#">auto_gcroot::get</a>	Gets the contained object.
<a href="#">auto_gcroot::release</a>	Releases the object from <code>auto_gcroot</code> management.
<a href="#">auto_gcroot::reset</a>	Destroy the current owned object and optionally take possession of a new object.
<a href="#">auto_gcroot::swap</a>	Swaps objects with another <code>auto_gcroot</code> .

### Public operators

NAME	DESCRIPTION
<a href="#">auto_gcroot::operator-&gt;</a>	The member access operator.
<a href="#">auto_gcroot::operator=</a>	Assignment operator.

NAME	DESCRIPTION
auto_gcroot::operator auto_gcroot	Type-cast operator between <code>auto_gcroot</code> and compatible types.
auto_gcroot::operator bool	Operator for using <code>auto_gcroot</code> in a conditional expression.
auto_gcroot::operator!	Operator for using <code>auto_gcroot</code> in a conditional expression.

## Requirements

Header file `<msclr\auto_gcroot.h>`

Namespace `msclr`

### auto\_gcroot::auto\_gcroot

The `auto_gcroot` constructor.

```
auto_gcroot(
    _element_type _ptr = nullptr
);
auto_gcroot(
    auto_gcroot<_element_type> & _right
);
template<typename _other_type>
auto_gcroot(
    auto_gcroot<_other_type> & _right
);
```

#### Parameters

`_ptr`

The object to own.

`_right`

An existing `auto_gcroot`.

#### Remarks

When constructing an `auto_gcroot` from an existing `auto_gcroot`, the existing `auto_gcroot` releases its object before transferring ownership of the object to the new `auto_gcroot`.

#### Example

```
// ms1_auto_gcroot_auto_gcroot.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class RefClassA {
protected:
    String^ m_s;
public:
    RefClassA(String^ s) : m_s(s) {
        Console::WriteLine( "in RefClassA constructor: " + m_s );
    }
}
```

```

}

~RefClassA() {
    Console::WriteLine( "in RefClassA destructor: " + m_s );
}

virtual void PrintHello() {
    Console::WriteLine( "Hello from {0} A!", m_s );
}

};

ref class RefClassB : RefClassA {
public:
    RefClassB( String^ s ) : RefClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

class ClassA { //unmanaged class
private:
    auto_gcroot<RefClassA^> m_a;

public:
    ClassA() : m_a( gcnew RefClassA( "unmanaged" ) ) {}
    ~ClassA() {} //no need to delete m_a

    void DoSomething() {
        m_a->PrintHello();
    }
};

int main()
{
{
    ClassA a;
    a.DoSomething();
} // a.m_a is automatically destroyed as a goes out of scope

{
    auto_gcroot<RefClassA^> a(gcnew RefClassA( "first" ) );
    a->PrintHello();
}

{
    auto_gcroot<RefClassB^> b(gcnew RefClassB( "second" ) );
    b->PrintHello();
    auto_gcroot<RefClassA^> a(b); //construct from derived type
    a->PrintHello();
    auto_gcroot<RefClassA^> a2(a); //construct from same type
    a2->PrintHello();
}

    Console::WriteLine("done");
}

```

```
in RefClassA constructor: unmanaged
Hello from unmanaged A!
in RefClassA destructor: unmanaged
in RefClassA constructor: first
Hello from first A!
in RefClassA destructor: first
in RefClassA constructor: second
Hello from second B!
Hello from second A!
Hello from second A!
in RefClassA destructor: second
done
```

## auto\_gcroot::~auto\_gcroot

The `auto_gcroot` destructor.

```
~auto_gcroot();
```

### Remarks

The destructor also destructs the owned object.

### Example

```
// msl_auto_gcroot_dtor.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
public:
    ClassA() { Console::WriteLine( "ClassA constructor" ); }
    ~ClassA() { Console::WriteLine( "ClassA destructor" ); }
};

int main()
{
    // create a new scope for a:
    {
        auto_gcroot<ClassA^> a = gcnew ClassA;
    }
    // a goes out of scope here, invoking its destructor
    // which in turns destructs the ClassA object.

    Console::WriteLine( "done" );
}
```

```
ClassA constructor
ClassA destructor
done
```

## auto\_gcroot::attach

Attach `auto_gcroot` to an object.

```
auto_gcroot<_element_type> & attach(
    _element_type _right
);
auto_gcroot<_element_type> & attach(
    auto_gcroot<_element_type> & _right
);
template<typename _other_type>
auto_gcroot<_element_type> & attach(
    auto_gcroot<_other_type> & _right
);
```

## Parameters

### *\_right*

The object to attach, or an `auto_gcroot` containing the object to attach.

## Return value

The current `auto_gcroot`.

## Remarks

If `_right` is an `auto_gcroot`, it releases ownership of its object before the object is attached to the current `auto_gcroot`.

## Example

```

// ms1_auto_gcroot_attach.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "in ClassA constructor:" + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor:" + m_s );
    }

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String ^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main() {
    auto_gcroot<ClassA^> a( gcnew ClassA( "first" ) );
    a->PrintHello();
    a.attach( gcnew ClassA( "second" ) ); // attach same type
    a->PrintHello();
    ClassA^ ha = gcnew ClassA( "third" );
    a.attach( ha ); // attach raw handle
    a->PrintHello();
    auto_gcroot<ClassB^> b( gcnew ClassB("fourth") );
    b->PrintHello();
    a.attach( b ); // attach derived type
    a->PrintHello();
}

```

```

in ClassA constructor:first
Hello from first A!
in ClassA constructor:second
in ClassA destructor:first
Hello from second A!
in ClassA constructor:third
in ClassA destructor:second
Hello from third A!
in ClassA constructor:fourth
Hello from fourth B!
in ClassA destructor:third
Hello from fourth A!
in ClassA destructor:fourth

```

## auto\_gcroot::get

Gets the contained object.

```
_element_type get() const;
```

## Return value

The contained object.

## Example

```
// msl_auto_gcroot_get.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ){
        Console::WriteLine( "in ClassA constructor:" + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor:" + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

void PrintA( ClassA^ a ) {
    a->PrintHello();
}

int main() {
    auto_gcroot<ClassA^> a = gcnew ClassA( "first" );
    a->PrintHello();

    ClassA^ a2 = a.get();
    a2->PrintHello();

    PrintA( a.get() );
}
```

```
in ClassA constructor:first
Hello from first A!
Hello from first A!
Hello from first A!
in ClassA destructor:first
```

## auto\_gcroot::release

Releases the object from `auto_gcroot` management.

```
_element_type release();
```

## Return value

The released object.

## Example

```

// ms1_auto_gcroot_release.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "ClassA destructor: " + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

int main()
{
    ClassA^ a;

    // create a new scope:
    {
        auto_gcroot<ClassA^> agc1 = gcnew ClassA( "first" );
        auto_gcroot<ClassA^> agc2 = gcnew ClassA( "second" );
        a = agc1.release();
    }
    // agc1 and agc2 go out of scope here

    a->PrintHello();

    Console::WriteLine( "done" );
}

```

```

ClassA constructor: first
ClassA constructor: second
ClassA destructor: second
Hello from first A!
done

```

## auto\_gcroot::reset

Destroy the current owned object and optionally take possession of a new object.

```

void reset(
    _element_type _new_ptr = nullptr
);

```

### Parameters

*\_new\_ptr*

(Optional) The new object.

### Example

```

// ms1_auto_gcroot_reset.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "ClassA destructor: " + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!" , m_s );
    }
};

int main()
{
    auto_gcroot<ClassA^> agc1 = gcnew ClassA( "first" );
    agc1->PrintHello();

    ClassA^ ha = gcnew ClassA( "second" );
    agc1.reset( ha ); // release first object, reference second
    agc1->PrintHello();

    agc1.reset(); // release second object, set to nullptr

    Console::WriteLine( "done" );
}

```

```

ClassA constructor: first
Hello from first A!
ClassA constructor: second
ClassA destructor: first
Hello from second A!
ClassA destructor: second
done

```

## auto\_gcroot::swap

Swaps objects with another `auto_gcroot`.

```

void swap(
    auto_gcroot<_element_type> & _right
);

```

### Parameters

`_right`

The `auto_gcroot` with which to swap objects.

### Example

```

// ms1_auto_gcroot_swap.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

int main() {
    auto_gcroot<String^> s1 = "string one";
    auto_gcroot<String^> s2 = "string two";

    Console::WriteLine( "s1 = '{0}', s2 = '{1}'", 
        s1->ToString(), s2->ToString() );
    s1.swap( s2 );
    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
}

```

```

s1 = 'string one', s2 = 'string two'
s1 = 'string two', s2 = 'string one'

```

## auto\_gcroot::operator->

The member access operator.

```

_element_type operator->() const;

```

### Return value

The object that's wrapped by `auto_gcroot`.

### Example

```

// ms1_auto_gcroot_op_arrow.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {}

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }

    int m_i;
};

int main() {
    auto_gcroot<ClassA^> a( gcnew ClassA( "first" ) );
    a->PrintHello();

    a->m_i = 5;
    Console::WriteLine( "a->m_i = {0}", a->m_i );
}

```

```
Hello from first A!
a->m_i = 5
```

## auto\_gcroot::operator=

Assignment operator.

```
auto_gcroot<_element_type> & operator=(
    _element_type _right
);
auto_gcroot<_element_type> & operator=(
    auto_gcroot<_element_type> & _right
);
template<typename _other_type>
auto_gcroot<_element_type> & operator=(
    auto_gcroot<_other_type> & _right
);
```

### Parameters

*\_right*

The object or `auto_gcroot` to be assigned to the current `auto_gcroot`.

### Return value

The current `auto_gcroot`, now owning `_right`.

### Example

```

// ms1_auto_gcroot_operator_equals.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA(String^ s) : m_s(s) {
        Console::WriteLine( "in ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor: " + m_s );
    }

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main()
{
    auto_gcroot<ClassA^> a;
    auto_gcroot<ClassA^> a2(gcnew ClassA( "first" ) );
    a = a2; // assign from same type
    a->PrintHello();

    ClassA^ ha = gcnew ClassA( "second" );
    a = ha; // assign from raw handle

    auto_gcroot<ClassB^> b(gcnew ClassB( "third" ) );
    b->PrintHello();
    a = b; // assign from derived type
    a->PrintHello();

    Console::WriteLine("done");
}

```

```

in ClassA constructor: first
Hello from first A!
in ClassA constructor: second
in ClassA destructor: first
in ClassA constructor: third
Hello from third B!
in ClassA destructor: second
Hello from third A!
done
in ClassA destructor: third

```

## auto\_gcroot::operator auto\_gcroot

Type-cast operator between `auto_gcroot` and compatible types.

```
template<typename _other_type>
operator auto_gcroot<_other_type>();
```

## Return value

The current `auto_gcroot` cast to `auto_gcroot<_other_type>`.

## Example

```
// msl_auto_gcroot_op_auto_gcroot.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {}

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String ^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main() {
    auto_gcroot<ClassB^> b = gcnew ClassB("first");
    b->PrintHello();
    auto_gcroot<ClassA^> a = (auto_gcroot<ClassA^>)b;
    a->PrintHello();
}
```

```
Hello from first B!
Hello from first A!
```

## auto\_gcroot::operator bool

Operator for using `auto_gcroot` in a conditional expression.

```
operator bool() const;
```

## Return value

`true` if the wrapped object is valid; `false` otherwise.

## Remarks

This operator actually converts to `_detail_class::_safe_bool`, which is safer than `bool` because it can't be converted to an integral type.

## Example

```
// msl_auto_gcroot_operator_bool.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

int main() {
    auto_gcroot<String^> s;
    if ( s ) Console::WriteLine( "s is valid" );
    if ( !s ) Console::WriteLine( "s is invalid" );
    s = "something";
    if ( s ) Console::WriteLine( "now s is valid" );
    if ( !s ) Console::WriteLine( "now s is invalid" );
    s.reset();
    if ( s ) Console::WriteLine( "now s is valid" );
    if ( !s ) Console::WriteLine( "now s is invalid" );
}
```

```
s is invalid
now s is valid
now s is invalid
```

## auto\_gcroot::operator!

Operator for using `auto_gcroot` in a conditional expression.

```
bool operator!() const;
```

### Return value

`true` if the wrapped object is invalid; `false` otherwise.

### Example

```
// msl_auto_gcroot_operator_not.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

int main() {
    auto_gcroot<String^> s;
    if ( s ) Console::WriteLine( "s is valid" );
    if ( !s ) Console::WriteLine( "s is invalid" );
    s = "something";
    if ( s ) Console::WriteLine( "now s is valid" );
    if ( !s ) Console::WriteLine( "now s is invalid" );
    s.reset();
    if ( s ) Console::WriteLine( "now s is valid" );
    if ( !s ) Console::WriteLine( "now s is invalid" );
}
```

```
s is invalid
now s is valid
now s is invalid
```

# swap Function (auto\_gcroot)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Swaps objects between one `auto_gcroot` and another.

## Syntax

```
template<typename _element_type>
void swap(
    auto_gcroot<_element_type> & _left,
    auto_gcroot<_element_type> & _right
);
```

### Parameters

`_left`

An `auto_gcroot`.

`_right`

Another `auto_gcroot`.

## Example

```
// msl_swap_auto_gcroot.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

int main() {
    auto_gcroot<String^> s1 = "string one";
    auto_gcroot<String^> s2 = "string two";

    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
    swap( s1, s2 );
    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
}
```

```
s1 = 'string one', s2 = 'string two'
s1 = 'string two', s2 = 'string one'
```

## Requirements

**Header file** `<msclr\auto_gcroot.h>`

**Namespace** `msclr`

## See also

[auto\\_gcroot](#)

auto\_gcroot::swap

# auto\_handle

9/20/2022 • 2 minutes to read • [Edit Online](#)

Defines the `auto_handle` class and `swap` function.

## Syntax

```
#include <msclr\auto_handle.h>
```

## Remarks

In this header file:

[auto\\_handle Class](#)

[swap Function \(auto\\_handle\)](#)

## See also

[C++ Support Library](#)

# auto\_handle Class

9/20/2022 • 7 minutes to read • [Edit Online](#)

Automatic resource management, which can be used to embed a virtual handle into a managed type.

## Syntax

```
template<typename _element_type>
ref class auto_handle;
```

### Parameters

*\_element\_type*

The managed type to be embedded.

## Members

### Public constructors

NAME	DESCRIPTION
<code>auto_handle::auto_handle</code>	The <code>auto_handle</code> constructor.
<code>auto_handle::~auto_handle</code>	The <code>auto_handle</code> destructor.

### Public methods

NAME	DESCRIPTION
<code>auto_handle::get</code>	Gets the contained object.
<code>auto_handle::release</code>	Releases the object from <code>auto_handle</code> management.
<code>auto_handle::reset</code>	Destroy the current owned object and optionally take possession of a new object.
<code>auto_handle::swap</code>	Swaps objects with another <code>auto_handle</code> .

### Public operators

NAME	DESCRIPTION
<code>auto_handle::operator-&gt;</code>	The member access operator.
<code>auto_handle::operator=</code>	Assignment operator.
<code>auto_handle::operator auto_handle</code>	Type-cast operator between <code>auto_handle</code> and compatible types.

NAME	DESCRIPTION
<code>auto_handle::operator bool</code>	Operator for using <code>auto_handle</code> in a conditional expression.
<code>auto_handle::operator!</code>	Operator for using <code>auto_handle</code> in a conditional expression.

## Requirements

Header file `<msclr\auto_handle.h>`

Namespace `msclr`

### `auto_handle::auto_handle`

The `auto_handle` constructor.

```
auto_handle();
auto_handle(
    _element_type ^ _ptr
);
auto_handle(
    auto_handle<_element_type> % _right
);
template<typename _other_type>
auto_handle(
    auto_handle<_other_type> % _right
);
```

#### Parameters

`_ptr`

The object to own.

`_right`

An existing `auto_handle`.

#### Example

```

// ms1_auto_handle_auto_handle.cpp
// compile with: /clr
#include "msclr\auto_handle.h"

using namespace System;
using namespace msclr;
ref class RefClassA {
protected:
    String^ m_s;
public:
    RefClassA(String^ s) : m_s(s) {
        Console::WriteLine( "in RefClassA constructor: " + m_s );
    }
    ~RefClassA() {
        Console::WriteLine( "in RefClassA destructor: " + m_s );
    }

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!" , m_s );
    }
};

ref class RefClassB : RefClassA {
public:
    RefClassB( String^ s ) : RefClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!" , m_s );
    }
};

int main()
{
{
    auto_handle<RefClassA> a(gcnew RefClassA( "first" ) );
    a->PrintHello();
}

{
    auto_handle<RefClassB> b(gcnew RefClassB( "second" ) );
    b->PrintHello();
    auto_handle<RefClassA> a(b); //construct from derived type
    a->PrintHello();
    auto_handle<RefClassA> a2(a); //construct from same type
    a2->PrintHello();
}

    Console::WriteLine("done");
}

```

```

in RefClassA constructor: first
Hello from first A!
in RefClassA destructor: first
in RefClassA constructor: second
Hello from second B!
Hello from second A!
Hello from second A!
in RefClassA destructor: second
done

```

## auto\_handle::~auto\_handle

The `auto_handle` destructor.

```
~auto_handle();
```

## Remarks

The destructor also destructs the owned object.

## Example

```
// msl_auto_handle_dtor.cpp
// compile with: /clr
#include "msclr\auto_handle.h"

using namespace System;
using namespace msclr;

ref class ClassA {
public:
    ClassA() { Console::WriteLine( "ClassA constructor" ); }
    ~ClassA() { Console::WriteLine( "ClassA destructor" ); }
};

int main()
{
    // create a new scope for a:
    {
        auto_handle<ClassA> a = gcnew ClassA;
    }
    // a goes out of scope here, invoking its destructor
    // which in turns destructs the ClassA object.

    Console::WriteLine( "done" );
}
```

```
ClassA constructor
ClassA destructor
done
```

## auto\_handle::get

Gets the contained object.

```
_element_type ^ get();
```

## Return value

The contained object.

## Example

```

// ms1_auto_handle_get.cpp
// compile with: /clr
#include "msclr\auto_handle.h"

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ){
        Console::WriteLine( "in ClassA constructor:" + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor:" + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

void PrintA( ClassA^ a ) {
    a->PrintHello();
}

int main() {
    auto_handle<ClassA> a = gcnew ClassA( "first" );
    a->PrintHello();

    ClassA^ a2 = a.get();
    a2->PrintHello();

    PrintA( a.get() );
}

```

```

in ClassA constructor:first
Hello from first A!
Hello from first A!
Hello from first A!
in ClassA destructor:first

```

## auto\_handle::release

Releases the object from `auto_handle` management.

```

_element_type ^ release();

```

### Return value

The released object.

### Example

```

// ms1_auto_handle_release.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "ClassA destructor: " + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

int main()
{
    ClassA^ a;

    // create a new scope:
    {
        auto_handle<ClassA> agc1 = gcnew ClassA( "first" );
        auto_handle<ClassA> agc2 = gcnew ClassA( "second" );
        a = agc1.release();
    }
    // agc1 and agc2 go out of scope here

    a->PrintHello();

    Console::WriteLine( "done" );
}

```

```

ClassA constructor: first
ClassA constructor: second
ClassA destructor: second
Hello from first A!
done

```

## auto\_handle::reset

Destroy the current owned object and optionally take possession of a new object.

```

void reset(
    _element_type ^ _new_ptr
);
void reset();

```

### Parameters

*\_new\_ptr*

(Optional) The new object.

### Example

```

// ms1_auto_handle_reset.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "ClassA destructor: " + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

int main()
{
    auto_handle<ClassA> agc1 = gcnew ClassA( "first" );
    agc1->PrintHello();

    ClassA^ ha = gcnew ClassA( "second" );
    agc1.reset( ha ); // release first object, reference second
    agc1->PrintHello();

    agc1.reset(); // release second object, set to nullptr

    Console::WriteLine( "done" );
}

```

```

ClassA constructor: first
Hello from first A!
ClassA constructor: second
ClassA destructor: first
Hello from second A!
ClassA destructor: second
done

```

## auto\_handle::swap

Swaps objects with another `auto_handle`.

```

void swap(
    auto_handle<_element_type> % _right
);

```

### Parameters

`_right`

The `auto_handle` with which to swap objects.

### Example

```

// ms1_auto_handle_swap.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

int main() {
    auto_handle<String> s1 = "string one";
    auto_handle<String> s2 = "string two";

    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
    s1.swap( s2 );
    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
}

```

```

s1 = 'string one', s2 = 'string two'
s1 = 'string two', s2 = 'string one'

```

## auto\_handle::operator->

The member access operator.

```

_element_type ^ operator->();

```

### Return value

The object that's wrapped by `auto_handle`.

### Example

```

// ms1_auto_handle_op_arrow.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {}

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }

    int m_i;
};

int main() {
    auto_handle<ClassA> a( gcnew ClassA( "first" ) );
    a->PrintHello();

    a->m_i = 5;
    Console::WriteLine( "a->m_i = {0}", a->m_i );
}

```

```
Hello from first A!
a->m_i = 5
```

## auto\_handle::operator=

Assignment operator.

```
auto_handle<_element_type>% operator=(  
    auto_handle<_element_type>% _right  
>;  
template<typename _other_type>  
auto_handle<_element_type>% operator=(  
    auto_handle<_other_type>% _right  
>;
```

### Parameters

*\_right*

The `auto_handle` to be assigned to the current `auto_handle`.

### Return value

The current `auto_handle`, now owning `_right`.

### Example

```

// ms1_auto_handle_op_assign.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA(String^ s) : m_s(s) {
        Console::WriteLine( "in ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor: " + m_s );
    }

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main()
{
    auto_handle<ClassA> a;
    auto_handle<ClassA> a2(gcnew ClassA( "first" ) );
    a = a2; // assign from same type
    a->PrintHello();

    auto_handle<ClassB> b(gcnew ClassB( "second" ) );
    b->PrintHello();
    a = b; // assign from derived type
    a->PrintHello();

    Console::WriteLine("done");
}

```

```

in ClassA constructor: first
Hello from first A!
in ClassA constructor: second
Hello from second B!
in ClassA destructor: first
Hello from second A!
done
in ClassA destructor: second

```

## auto\_handle::operator auto\_handle

Type-cast operator between `auto_handle` and compatible types.

```

template<typename _other_type>
operator auto_handle<_other_type>();

```

## Return value

The current `auto_handle` cast to `auto_handle<_other_type>`.

## Example

```
// msl_auto_handle_op_auto_handle.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {}

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String ^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main() {
    auto_handle<ClassB> b = gcnew ClassB("first");
    b->PrintHello();
    auto_handle<ClassA> a = (auto_handle<ClassA>)b;
    a->PrintHello();
}
```

```
Hello from first B!
Hello from first A!
```

## auto\_handle::operator bool

Operator for using `auto_handle` in a conditional expression.

```
operator bool();
```

## Return value

`true` if the wrapped object is valid; `false` otherwise.

## Remarks

This operator actually converts to `_detail_class::_safe_bool` which is safer than `bool` because it can't be converted to an integral type.

## Example

```
// msl_auto_handle_operator_bool.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

int main() {
    auto_handle<String> s1;
    auto_handle<String> s2 = "hi";
    if ( s1 ) Console::WriteLine( "s1 is valid" );
    if ( !s1 ) Console::WriteLine( "s1 is invalid" );
    if ( s2 ) Console::WriteLine( "s2 is valid" );
    if ( !s2 ) Console::WriteLine( "s2 is invalid" );
    s2.reset();
    if ( s2 ) Console::WriteLine( "s2 is now valid" );
    if ( !s2 ) Console::WriteLine( "s2 is now invalid" );
}
```

```
s1 is invalid
s2 is valid
s2 is now invalid
```

## auto\_handle::operator!

Operator for using `auto_handle` in a conditional expression.

```
bool operator!();
```

### Return value

`true` if the wrapped object is invalid; `false` otherwise.

### Example

```
// msl_auto_handle_operator_not.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

int main() {
    auto_handle<String> s1;
    auto_handle<String> s2 = "something";
    if ( s1 ) Console::WriteLine( "s1 is valid" );
    if ( !s1 ) Console::WriteLine( "s1 is invalid" );
    if ( s2 ) Console::WriteLine( "s2 is valid" );
    if ( !s2 ) Console::WriteLine( "s2 is invalid" );
    s2.reset();
    if ( s2 ) Console::WriteLine( "s2 is now valid" );
    if ( !s2 ) Console::WriteLine( "s2 is now invalid" );
}
```

```
s1 is invalid
s2 is valid
s2 is now invalid
```

# swap Function (auto\_handle)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Swaps objects between one `auto_handle` and another.

## Syntax

```
template<typename _element_type>
void swap(
    auto_handle<_element_type>% _left,
    auto_handle<_element_type>% _right
);
```

### Parameters

`_left`

An `auto_handle`.

`_right`

Another `auto_handle`.

## Example

```
// msl_swap_auto_handle.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

int main() {
    auto_handle<String> s1 = "string one";
    auto_handle<String> s2 = "string two";

    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
    swap( s1, s2 );
    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
}
```

```
s1 = 'string one', s2 = 'string two'
s1 = 'string two', s2 = 'string one'
```

## Requirements

**Header file** `<msclr\auto_handle.h>`

**Namespace** `msclr`

## See also

[auto\\_handle](#)

auto\_handle::swap

# Synchronization (lock Class)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Provides a mechanism to automate taking a lock that synchronizes access to an object.

## In This Section

[lock](#)

## See also

[C++ Support Library](#)

# lock

9/20/2022 • 2 minutes to read • [Edit Online](#)

Defines the `lock` class that automates synchronization of access to an object.

## Syntax

```
#include <msclr\lock.h>
```

## Remarks

In this header file:

[lock Class](#)

[lock\\_when Enum](#)

## See also

[C++ Support Library](#)

# lock Class

9/20/2022 • 16 minutes to read • [Edit Online](#)

This class automates taking a lock for synchronizing access to an object from several threads. When constructed it acquires the lock and when destroyed it releases the lock.

## Syntax

```
ref class lock;
```

## Remarks

`lock` is available only for CLR objects and can only be used in CLR code.

Internally, the lock class uses [Monitor](#) to synchronize access. For more information, see the referenced article.

## Members

### Public constructors

NAME	DESCRIPTION
<code>lock::lock</code>	Constructs a <code>lock</code> object, optionally waiting to acquire the lock forever, for a specified amount of time, or not at all.
<code>lock::~lock</code>	Destructs a <code>lock</code> object.

### Public methods

NAME	DESCRIPTION
<code>lock::acquire</code>	Acquires a lock on an object, optionally waiting to acquire the lock forever, for a specified amount of time, or not at all.
<code>lock::is_locked</code>	Indicates whether a lock is being held.
<code>lock::release</code>	Releases a lock.
<code>lock::try_acquire</code>	Acquires a lock on an object, waiting for a specified amount of time and returning a <code>bool</code> to report the success of acquisition instead of throwing an exception.

### Public operators

NAME	DESCRIPTION
<code>lock::operator bool</code>	Operator for using <code>lock</code> in a conditional expression.
<code>lock::operator==</code>	Equality operator.

NAME	DESCRIPTION
<code>lock::operator!=</code>	Inequality operator.

## Requirements

Header file <msclr\lock.h>

Namespace msclr

### lock::lock

Constructs a `lock` object, optionally waiting to acquire the lock forever, for a specified amount of time, or not at all.

```
template<class T> lock(
    T ^ _object
);
template<class T> lock(
    T ^ _object,
    int _timeout
);
template<class T> lock(
    T ^ _object,
    System::TimeSpan _timeout
);
template<class T> lock(
    T ^ _object,
    lock_later
);
```

#### Parameters

`_object`

The object to be locked.

`_timeout`

Time out value in milliseconds or as a [TimeSpan](#).

#### Exceptions

Throws [ApplicationException](#) if lock acquisition doesn't occur before timeout.

#### Remarks

The first three forms of the constructor try to acquire a lock on `_object` within the specified timeout period (or [Infinite](#) if none is specified).

The fourth form of the constructor doesn't acquire a lock on `_object`. `lock_later` is a member of the `lock_when` enum. Use `lock::acquire` or `lock::try_acquire` to acquire the lock in this case.

The lock will automatically be released when the destructor is called.

`_object` can't be [ReaderWriterLock](#). If it is, a compiler error will result.

#### Example

This example uses a single instance of a class across several threads. The class uses a lock on itself to make sure that accesses to its internal data are consistent for each thread. The main application thread uses a lock on the same instance of the class to periodically check to see if any worker threads still exist. The main application then waits to exit until all worker threads have completed their tasks.

```

// ms1_lock_lock.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                               Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                               Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}

```

```
}
```

```
In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.
```

## lock::~lock

Destructs a `lock` object.

```
~lock();
```

### Remarks

The destructor calls `lock::release`.

### Example

This example uses a single instance of a class across several threads. The class uses a lock on itself to make sure that accesses to its internal data are consistent for each thread. The main application thread uses a lock on the same instance of the class to periodically check to see if any worker threads still exist. The main application then waits to exit until all worker threads have completed their tasks.

```
// ms1_lock_dtor.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);
        }
    }
}
```

```

        Counter = 0;
        // lock is automatically released when it goes out of scope and its destructor is called
    }
    catch (...) {
        Console::WriteLine("Couldn't acquire lock!");
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

## lock::acquire

Acquires a lock on an object, optionally waiting to acquire the lock forever, for a specified amount of time, or not at all.

```
void acquire();
void acquire(
    int _timeout
);
void acquire(
    System::TimeSpan _timeout
);
```

## Parameters

*\_timeout*

Timeout value in milliseconds or as a [TimeSpan](#).

## Exceptions

Throws [ApplicationException](#) if lock acquisition doesn't occur before timeout.

## Remarks

If a timeout value isn't supplied, the default timeout is [Infinite](#).

If a lock has already been acquired, this function does nothing.

## Example

This example uses a single instance of a class across several threads. The class uses a lock on itself to make sure that accesses to its internal data are consistent for each thread. The main application thread uses a lock on the same instance of the class to periodically check to see if any worker threads still exist. The main application then waits to exit until all worker threads have completed their tasks.

```
// msl_lock_acquire.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {

```

```

        Console::WriteLine("Couldn't acquire lock!");
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

## lock::is\_locked

Indicates whether a lock is being held.

```
bool is_locked();
```

### Return value

`true` if a lock is held, `false` otherwise.

### Example

This example uses a single instance of a class across several threads. The class uses a lock on itself to make sure that accesses to its internal data are consistent for each thread. The main application thread uses a lock on the same instance of the class to periodically check to see if any worker threads still exist, and waits to exit until all worker threads have completed their tasks.

```

// msl_lock_is_locked.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                               Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                               Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        l.try_acquire(50); // try to acquire lock, don't throw an exception if can't
        if (l.is_locked()) { // check if we got the lock
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}

```

```
}
```

```
In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.
```

## lock::operator bool

Operator for using `lock` in a conditional expression.

```
operator bool();
```

### Return value

`true` if a lock is held, `false` otherwise.

### Remarks

This operator actually converts to `_detail_class::_safe_bool` which is safer than `bool` because it can't be converted to an integral type.

### Example

This example uses a single instance of a class across several threads. The class uses a lock on itself to make sure that accesses to its internal data are consistent for each thread. The main application thread uses a lock on the same instance of the class to periodically check to see if any worker threads still exist. The main application waits to exit until all worker threads have completed their tasks.

```
// msl_lock_op_bool.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {

```

```

        Counter++;
        Thread::Sleep(10);
    }

    Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                      Counter);

    Counter = 0;
    // lock is automatically released when it goes out of scope and its destructor is called
}
catch (...) {
    Console::WriteLine("Couldn't acquire lock!");
}

ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        l.try_acquire(50); // try to acquire lock, don't throw an exception if can't
        if (l) { // use bool operator to check for lock
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

## lock::release

Releases a lock.

```
void release();
```

## Remarks

If no lock is being held, `release` does nothing.

You don't have to call this function explicitly. When a `lock` object goes out of scope, its destructor calls `release`.

## Example

This example uses a single instance of a class across several threads. The class uses a lock on itself to make sure that accesses to its internal data are consistent for each thread. The main application thread uses a lock on the same instance of the class to periodically check to see if any worker threads still exist. The main application then waits to exit until all worker threads have completed their tasks.

```
// msl_lock_release.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                               Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                               Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }
        ThreadCount--;
    }
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
}
```

```

// KEEP OUR MAIN THREAD ALIVE UNTIL ALL WORKER THREADS HAVE COMPLETED
lock l(cc, lock_later); // don't lock now, just create the object
while (true) {
    if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
        if (0 == cc->ThreadCount) {
            Console::WriteLine("All threads completed.");
            break; // all threads are gone, exit while
        }
        else {
            Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
            l.release(); // some threads exist, let them do their work
        }
    }
}
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

## lock::try\_acquire

Acquires a lock on an object, waiting for a specified amount of time and returning a `bool` to report the success of acquisition instead of throwing an exception.

```

bool try_acquire(
    int _timeout_ms
);
bool try_acquire(
    System::TimeSpan _timeout
);

```

### Parameters

`_timeout`

Timeout value in milliseconds or as a [TimeSpan](#).

### Return value

`true` if lock was acquired, `false` otherwise.

### Remarks

If a lock has already been acquired, this function does nothing.

### Example

This example uses a single instance of a class across several threads. The class uses a lock on itself to make sure that accesses to its internal data are consistent for each thread. The main application thread uses a lock on the same instance of the class to periodically check to see if any worker threads still exist. The main application then waits to exit until all worker threads have completed their tasks.

```

// msl_lock_try_acquire.cpp
// compile with: /clr
#include <msclr/lock.h>

```

```

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

```
In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.
```

## lock::operator==

Equality operator.

```
template<class T> bool operator==(  
    T t  
>;
```

### Parameters

*t*

The object to compare for equality.

### Return value

Returns `true` if *t* is the same as the lock's object, `false` otherwise.

### Example

```
// msl_lock_op_eq.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

int main () {
    Object^ o1 = gcnew Object;
    lock l1(o1);
    if (l1 == o1) {
        Console::WriteLine("Equal!");
    }
}
```

```
Equal!
```

## lock::operator!=

Inequality operator.

```
template<class T> bool operator!=(  
    T t  
>;
```

### Parameters

*t*

The object to compare for inequality.

### Return value

Returns `true` if `t` differs from the lock's object, `false` otherwise.

### Example

```
// msl_lock_op_ineq.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

int main () {
    Object^ o1 = gcnew Object;
    Object^ o2 = gcnew Object;
    lock l1(o1);
    if (l1 != o2) {
        Console::WriteLine("Inequal!");
    }
}
```

```
Inequal!
```

# lock\_when Enum

9/20/2022 • 2 minutes to read • [Edit Online](#)

Specifies deferred locking.

## Syntax

```
enum lock_when {
    lock_later
};
```

## Remarks

When passed to `lock::lock`, `lock_later` specifies that the lock is not to be taken now.

## Example

This example uses a single instance of a class across multiple threads. The class uses a lock on itself to ensure that accesses to its internal data are consistent for each thread. The main application thread uses a lock on the same instance of the class to periodically check to see if any worker threads still exist, and waits to exit until all worker threads have completed their tasks.

```
// msl_lock_lock_when.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
    }
}
```

```

        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }

        ThreadCount--;
    }
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

## Requirements

Header file <msclr\lock.h>

Namespace msclr

## See also

[lock](#)

# Calling Functions in a Specific Application Domain

9/20/2022 • 2 minutes to read • [Edit Online](#)

Supports calling functions in a specific application domain.

## In This Section

[call\\_in\\_appdomain Function](#)

## See also

[C++ Support Library](#)

# call\_in\_appdomain Function

9/20/2022 • 2 minutes to read • [Edit Online](#)

Executes a function in a specified application domain.

## Syntax

```
template <typename ArgType1, ...typename ArgTypeN>
void call_in_appdomain(
    DWORD appdomainId,
    void (*voidFunc)(ArgType1, ...ArgTypeN) [ ,
        ArgType1 arg1,
        ...
        ArgTypeN argN ]
);
template <typename RetType, typename ArgType1, ...typename ArgTypeN>
RetType call_in_appdomain(
    DWORD appdomainId,
    RetType (*nonvoidFunc)(ArgType1, ...ArgTypeN) [ ,
        ArgType1 arg1,
        ...
        ArgTypeN argN ]
);
```

### Parameters

*appdomainId*

The appdomain in which to call the function.

*voidFunc*

Pointer to a `void` function that takes N parameters ( $0 \leq N \leq 15$ ).

*nonvoidFunc*

Pointer to a non-`void` function that takes N parameters ( $0 \leq N \leq 15$ ).

*arg1...argN*

Zero to 15 parameters to be passed to `voidFunc` or `nonvoidFunc` in the other appdomain.

## Return Value

The result of executing `voidFunc` or `nonvoidFunc` in the specified application domain.

## Remarks

The arguments of the function passed to `call_in_appdomain` must not be CLR types.

## Example

```

// msl_call_in_appdomain.cpp
// compile with: /clr

// Defines two functions: one takes a parameter and returns nothing,
// the other takes no parameters and returns an int. Calls both
// functions in the default appdomain and in a newly-created
// application domain using call_in_appdomain.

#include <msclr\appdomain.h>

using namespace System;
using namespace msclr;

void PrintCurrentDomainName( char* format )
{
    String^ s = gcnew String(format);
    Console::WriteLine( s, AppDomain::CurrentDomain->FriendlyName );
}

int GetDomainId()
{
    return AppDomain::CurrentDomain->Id;
}

int main()
{
    AppDomain^ appDomain1 = AppDomain::CreateDomain( "First Domain" );

    call_in_appdomain( AppDomain::CurrentDomain->Id,
                       &PrintCurrentDomainName,
                       (char*)"default appdomain: {0}" );
    call_in_appdomain( appDomain1->Id,
                       &PrintCurrentDomainName,
                       (char*)"in appDomain1: {0}" );

    int id;
    id = call_in_appdomain( AppDomain::CurrentDomain->Id, &GetDomainId );
    Console::WriteLine( "default appdomain id = {0}", id );
    id = call_in_appdomain( appDomain1->Id, &GetDomainId );
    Console::WriteLine( "appDomain1 id = {0}", id );
}

```

## Output

```

default appdomain: msl_call_in_appdomain.exe
in appDomain1: First Domain
default appdomain id = 1
appDomain1 id = 2

```

## Requirements

**Header file** <msclr\appdomain.h>

**Namespace** msclr

# com::ptr

9/20/2022 • 2 minutes to read • [Edit Online](#)

A wrapper for a COM object that can be used as a member of a CLR class. The wrapper also automates lifetime management of the COM object, releasing owned references on the object when its destructor is called. Analogous to [CComPtr Class](#).

## Syntax

```
#include <msclr\com\ptr.h>
```

## Remarks

[com::ptr Class](#) is defined in the `<msclr\com\ptr.h>` file.

## See also

[C++ Support Library](#)

# com::ptr Class

9/20/2022 • 24 minutes to read • [Edit Online](#)

A wrapper for a COM object that can be used as a member of a CLR class. The wrapper also automates lifetime management of the COM object, releasing all owned references on the object when its destructor is called. Analogous to [CComPtr class](#).

## Syntax

```
template<class _interface_type>
ref class ptr;
```

### Parameters

*\_interface\_type*

COM interface.

## Remarks

A `com::ptr` can also be used as a local function variable to simplify various COM tasks and to automate lifetime management.

A `com::ptr` can't be used directly as a function parameter; use a [Tracking reference operator](#) or a [Handle to object operator \(^\)](#) instead.

A `com::ptr` can't be directly returned from a function; use a handle instead.

## Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. Calling the public methods of the class results in calls to the contained `IXMLDOMDocument` object. The sample creates an instance of an XML document, fills it with some simple XML, and does a simplified walk of the nodes in the parsed document tree to print the XML to the console.

```
// comptr.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }
}
```

```

void LoadXml(String^ xml) {
    pin_ptr<const wchar_t> pinnedXml = PtrToStringChars(xml);
    BSTR bstr = NULL;

    try {
        // load some XML into the document
        bstr = ::SysAllocString(pinnedXml);
        if (NULL == bstr) {
            throw gcnew OutOfMemoryException;
        }
        VARIANT_BOOL bIsSuccessful = false;
        // use operator -> to call IXMODOMDocument member function
        Marshal::ThrowExceptionForHR(m_ptrDoc->loadXML(bstr, &bIsSuccessful));
    }
    finally {
        ::SysFreeString(bstr);
    }
}

// simplified function to write just the first xml node to the console
void WriteXml() {
    IXMLDOMNode* pNode = NULL;

    try {
        // the first child of the document is the first real xml node
        Marshal::ThrowExceptionForHR(m_ptrDoc->get_firstChild(&pNode));
        if (NULL != pNode) {
            WriteNode(pNode);
        }
    }
    finally {
        if (NULL != pNode) {
            pNode->Release();
        }
    }
}

// note that the destructor will call the com::ptr destructor
// and automatically release the reference to the COM object

private:
    // simplified function that only writes the node
    void WriteNode(IXMLDOMNode* pNode) {
        BSTR bstr = NULL;

        try {
            // write out the name and text properties
            Marshal::ThrowExceptionForHR(pNode->get_nodeName(&bstr));
            String^ strName = gcnew String(bstr);
            Console::Write("<{0}>", strName);
            ::SysFreeString(bstr);
            bstr = NULL;

            Marshal::ThrowExceptionForHR(pNode->get_text(&bstr));
            Console::Write(gcnew String(bstr));
            ::SysFreeString(bstr);
            bstr = NULL;

            Console::WriteLine("</{0}>", strName);
        }
        finally {
            ::SysFreeString(bstr);
        }
    }

    com::ptr<IXMODOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object

```

```
// add the .h file header to handle on the DOM document object
int main() {
    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // stream some xml into the document
        doc.LoadXml("<word>persnickety</word>");

        // write the document to the console
        doc.WriteXml();
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}
```

```
<word>persnickety</word>
```

## Members

### Public constructors

NAME	DESCRIPTION
ptr::ptr	Constructs a <code>com::ptr</code> to wrap a COM object.
ptr::~ptr	Destructs a <code>com::ptr</code> .

### Public methods

NAME	DESCRIPTION
ptr::Attach	Attaches a COM object to a <code>com::ptr</code> .
ptr::CreateInstance	Creates an instance of a COM object within a <code>com::ptr</code> .
ptr::Detach	Gives up ownership of the COM object, returning a pointer to the object.
ptr::GetInterface	Creates an instance of a COM object within a <code>com::ptr</code> .
ptr::QueryInterface	Queries the owned COM object for an interface and attaches the result to another <code>com::ptr</code> .
ptr::Release	Releases all owned references on the COM object.

### Public operators

NAME	DESCRIPTION
<code>ptr::operator-&gt;</code>	Member access operator, used to call methods on the owned COM object.
<code>ptr::operator=</code>	Attaches a COM object to a <code>com::ptr</code> .

NAME	DESCRIPTION
<code>ptr::operator bool</code>	Operator for using <code>com::ptr</code> in a conditional expression.
<code>ptr::operator!</code>	Operator to determine if the owned COM object is invalid.

## Requirements

**Header file** <msclr\com\ptr.h>

**Namespace** msclr::com

### ptr::ptr

Returns a pointer to the owned COM object.

```
ptr();
ptr(
    _interface_type * p
);
```

#### Parameters

*P*

A COM interface pointer.

#### Remarks

The no-argument constructor assigns `nullptr` to the underlying object handle. Future calls to the `com::ptr` will validate the internal object and silently fail until an object is created or attached.

The one-argument constructor adds a reference to the COM object but doesn't release the caller's reference, so the caller must call `Release` on the COM object to truly give up control. When the `com::ptr`'s destructor is called it will automatically release its references on the COM object.

Passing `NULL` to this constructor is the same as calling the no-argument version.

#### Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. It demonstrates usage of both versions of the constructor.

```

// comprtr_ptr.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // construct the internal com::ptr with a COM object
    XmlDocument(IXMLDOMDocument* pDoc) : m_ptrDoc(pDoc) {}

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create an XML DOM document object
        Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
            CLSCTX_ALL, IID_IXMLDOMDocument, (void**)&pDoc));
        // construct the ref class with the COM object
        XmlDocument doc1(pDoc);

        // or create the class from a progid string
        XmlDocument doc2("Msxml2.DOMDocument.3.0");
    }
    // doc1 and doc2 destructors are called when they go out of scope
    // and the internal com::ptr releases its reference to the COM object
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}

```

## ptr::~ptr

Destructs a `com::ptr`.

```
~ptr();
```

## Remarks

On destruction, the `com::ptr` releases all references it owns to its COM object. Assuming that there are no other references held to the COM object, the COM object will be deleted and its memory freed.

## Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. In the `main` function, the two  `XmlDocument` objects' destructors will be called when they go out of the scope of the `try` block, resulting in the underlying `com::ptr` destructor being called, releasing all owned references to the COM object.

```

// comprtr_dtor.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // construct the internal com::ptr with a COM object
    XmlDocument(IXMLDOMDocument* pDoc) : m_ptrDoc(pDoc) {}

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create an XML DOM document object
        Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
            CLSCTX_ALL, IID_IXMLDOMDocument, (void**)&pDoc));
        // construct the ref class with the COM object
        XmlDocument doc1(pDoc);

        // or create the class from a progid string
        XmlDocument doc2("Msxml2.DOMDocument.3.0");
    }
    // doc1 and doc2 destructors are called when they go out of scope
    // and the internal com::ptr releases its reference to the COM object
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}
}

```

## ptr::Attach

Attaches a COM object to a `com::ptr`.

```
void Attach(
    _interface_type * _right
);
```

## Parameters

### \_right

The COM interface pointer to attach.

## Exceptions

If the `com::ptr` already owns a reference to a COM object, `Attach` throws [InvalidOperationException](#).

## Remarks

A call to `Attach` references the COM object but doesn't release the caller's reference to it.

Passing `NULL` to `Attach` results in no action being taken.

## Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. The `ReplaceDocument` member function first calls `Release` on any previously owned object and then calls `Attach` to attach a new document object.

```
// comprtr_attach.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // replace currently held COM object with another one
    void ReplaceDocument(IXMLDOMDocument* pDoc) {
        // release current document object
        m_ptrDoc.Release();
        // attach the new document object
        m_ptrDoc.Attach(pDoc);
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that creates a raw XML DOM Document object
IXMLDOMDocument* CreateDocument() {
    IXMLDOMDocument* pDoc = NULL;
    Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
        CLSCTX_INPROC_SERVER, IID_IXMLDOMDocument, (void**)&pDoc));
```

```
    return pDoc;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // get another document object from unmanaged function and
        // store it in place of the one held by our ref class
        pDoc = CreateDocument();
        doc.ReplaceDocument(pDoc);
        // no further need for raw object reference
        pDoc->Release();
        pDoc = NULL;
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}
```

## ptr::CreateInstance

Creates an instance of a COM object within a `com::ptr`.

```
void CreateInstance(
    System::String ^ progid,
    LPUNKNOWN pouter,
    DWORD cls_context
);
void CreateInstance(
    System::String ^ progid,
    LPUNKNOWN pouter
);
void CreateInstance(
    System::String ^ progid
);
void CreateInstance(
    const wchar_t * progid,
    LPUNKNOWN pouter,
    DWORD cls_context
);
void CreateInstance(
    const wchar_t * progid,
    LPUNKNOWN pouter
);
void CreateInstance(
    const wchar_t * progid
);
void CreateInstance(
    REFCLSID rclsid,
    LPUNKNOWN pouter,
    DWORD cls_context
);
void CreateInstance(
    REFCLSID rclsid,
    LPUNKNOWN pouter
);
void CreateInstance(
    REFCLSID rclsid
);
```

## Parameters

### *progid*

A `ProgID` string.

### *pouter*

Pointer to the aggregate object's `IUnknown` interface (the controlling `IUnknown`). If `pouter` isn't specified, `NULL` is used.

### *cls\_context*

Context in which the code that manages the newly created object will run. The values are taken from the `CLSEX` enumeration. If `cls_context` isn't specified, the value `CLSEX_ALL` is used.

### *rclsid*

`CLSID` associated with the data and code that will be used to create the object.

## Exceptions

If the `com::ptr` already owns a reference to a COM object, `CreateInstance` throws `InvalidOperationException`.

This function calls `CoCreateInstance` and uses `ThrowExceptionForHR` to convert any error `HRESULT` to an appropriate exception.

## Remarks

`CreateInstance` uses `CoCreateInstance` to create a new instance of the specified object, identified either from a ProgID or a CLSID. The `com::ptr` references the newly created object and will automatically release all owned

references upon destruction.

## Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. The class constructors use two different forms of `CreateInstance` to create the document object either from a ProgID or from a CLSID plus a CLSCTX.

```
// comptr_createinstance.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }
    XmlDocument(REFCLSID clsid, DWORD clsctx) {
        m_ptrDoc.CreateInstance(clsid, NULL, clsctx);
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        // create the class from a progid string
        XmlDocument doc1("Msxml2.DOMDocument.3.0");

        // or from a clsid with specific CLSCTX
        XmlDocument doc2(CLSID_DOMDocument30, CLSCTX_INPROC_SERVER);
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}
```

## ptr::Detach

Gives up ownership of the COM object, returning a pointer to the object.

```
_interface_type * Detach();
```

### Return value

The pointer to the COM object.

If no object is owned, NULL is returned.

## Exceptions

Internally, `QueryInterface` is called on the owned COM object and any error `HRESULT` is converted to an exception by `ThrowExceptionForHR`.

## Remarks

`Detach` first adds a reference to the COM object on behalf of the caller and then releases all references owned by the `com::ptr`. The caller must ultimately release the returned object to destroy it.

## Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. The `DetachDocument` member function calls `Detach` to give up ownership of the COM object and return a pointer to the caller.

```
// comprtr_detach.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // detach the COM object and return it
    // this releases the internal reference to the object
    IXMLDOMDocument* DetachDocument() {
        return m_ptrDoc.Detach();
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that loads XML into a raw XML DOM Document object
HRESULT LoadXml(IXMLDOMDocument* pDoc, BSTR bstrXml) {
    HRESULT hr = S_OK;
    VARIANT_BOOL bSuccess;
    hr = pDoc->loadXML(bstrXml, &bSuccess);
    if (S_OK == hr && !bSuccess) {
        hr = E_FAIL;
    }
    return hr;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;
    BSTR bstrXml = NULL;
    +nw s
```

```

    // create the class from a progid string
    XmlDocument doc("Msxml2.DOMDocument.3.0");

    bstrXml = ::SysAllocString(L"<word>persnickety</word>");
    if (NULL == bstrXml) {
        throw gcnew OutOfMemoryException("bstrXml");
    }
    // detach the document object from the ref class
    pDoc = doc.DetachDocument();
    // use unmanaged function and raw object to load xml
    Marshal::ThrowExceptionForHR(LoadXml(pDoc, bstrXml));
    // release document object as the ref class no longer owns it
    pDoc->Release();
    pDoc = NULL;
}
catch (Exception^ e) {
    Console::WriteLine(e);
}
finally {
    if (NULL != pDoc) {
        pDoc->Release();
    }
}

}
}

```

## ptr::GetInterface

Returns a pointer to the owned COM object.

```
_interface_type * GetInterface();
```

### Return value

A pointer to the owned COM object.

### Exceptions

Internally, `QueryInterface` is called on the owned COM object and any error `HRESULT` is converted to an exception by `ThrowExceptionForHR`.

### Remarks

The `com::ptr` adds a reference to the COM object on the caller's behalf and also keeps its own reference on the COM object. The caller must ultimately release the reference on the returned object or it will never be destroyed.

### Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. The `GetDocument` member function uses `GetInterface` to return a pointer to the COM object.

```

// comprt_getinterface.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object

```

```

ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // add a reference to and return the COM object
    // but keep an internal reference to the object
    IXMLDOMDocument* GetDocument() {
        return m_ptrDoc.GetInterface();
    }

    // simplified function that only writes the first node
    void WriteDocument() {
        IXMLDOMNode* pNode = NULL;
        BSTR bstr = NULL;

        try {
            // use operator -> to call XML Doc member
            Marshal::ThrowExceptionForHR(m_ptrDoc->get.firstChild(&pNode));
            if (NULL != pNode) {
                // write out the xml
                Marshal::ThrowExceptionForHR(pNode->get_nodeName(&bstr));
                String^ strName = gcnew String(bstr);
                Console::Write("<{0}>", strName);
                ::SysFreeString(bstr);
                bstr = NULL;

                Marshal::ThrowExceptionForHR(pNode->get_text(&bstr));
                Console::Write(gcnew String(bstr));
                ::SysFreeString(bstr);
                bstr = NULL;

                Console::WriteLine("</{0}>", strName);
            }
        }
        finally {
            if (NULL != pNode) {
                pNode->Release();
            }
            ::SysFreeString(bstr);
        }
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object
}

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that loads XML into a raw XML DOM Document object
HRESULT LoadXml(IXMLDOMDocument* pDoc, BSTR bstrXml) {
    HRESULT hr = S_OK;
    VARIANT_BOOL bSuccess;
    hr = pDoc->loadXML(bstrXml, &bSuccess);
    if (S_OK == hr && !bSuccess) {
        hr = E_FAIL;
    }
    return hr;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;
    BSTR bstrXml = NULL;
}

```

```

try {
    // create the class from a progid string
    XmlDocument doc("Msxml2.DOMDocument.3.0");

    bstrXml = ::SysAllocString(L"<word>persnickety</word>");
    if (NULL == bstrXml) {
        throw gcnew OutOfMemoryException("bstrXml");
    }
    // detach the document object from the ref class
    pDoc = doc.GetDocument();
    // use unmanaged function and raw object to load xml
    Marshal::ThrowExceptionForHR(LoadXml(pDoc, bstrXml));
    // release reference to document object (but ref class still references it)
    pDoc->Release();
    pDoc = NULL;

    // call another function on the ref class
    doc.WriteDocument();
}
catch (Exception^ e) {
    Console::WriteLine(e);
}
finally {
    if (NULL != pDoc) {
        pDoc->Release();
    }
}

}

```

<word>persnickety</word>

## ptr::QueryInterface

Queries the owned COM object for an interface and attaches the result to another `com::ptr`.

```

template<class _other_type>
void QueryInterface(
    ptr<_other_type> % other
);

```

### Parameters

*other*

The `com::ptr` that will get the interface.

### Exceptions

Internally, `QueryInterface` is called on the owned COM object and any error `HRESULT` is converted to an exception by `ThrowExceptionForHR`.

### Remarks

Use this method to create a COM wrapper for a different interface of the COM object owned by the current wrapper. This method calls `QueryInterface` through the owned COM object to request a pointer to a specific interface of the COM object and attaches the returned interface pointer to the passed-in `com::ptr`.

### Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. The `WriteTopLevelNode` member function uses `QueryInterface` to fill a local `com::ptr` with an `IXMLDOMNode` and then passes the `com::ptr` (by tracking reference) to a private member function that writes the node's name and

text properties to the console.

```
// comprtr_queryinterface.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    void LoadXml(String^ xml) {
        pin_ptr<const wchar_t> pinnedXml = PtrToStringChars(xml);
        BSTR bstr = NULL;

        try {
            // load some XML into our document
            bstr = ::SysAllocString(pinnedXml);
            if (NULL == bstr) {
                throw gcnew OutOfMemoryException();
            }
            VARIANT_BOOL bIsSuccessful = false;
            // use operator -> to call IXMDOMDocument member function
            Marshal::ThrowExceptionForHR(m_ptrDoc->loadXML(bstr, &bIsSuccessful));
        }
        finally {
            ::SysFreeString(bstr);
        }
    }

    // write the top level node to the console
    void WriteTopLevelNode() {
        com::ptr<IXMLDOMNode> ptrNode;

        // query for the top level node interface
        m_ptrDoc.QueryInterface(ptrNode);
        WriteNode(ptrNode);
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    // simplified function that only writes the node
    void WriteNode(com::ptr<IXMLDOMNode> % node) {
        BSTR bstr = NULL;

        try {
            // write out the name and text properties
            Marshal::ThrowExceptionForHR(node->get_nodeName(&bstr));
            String^ strName = gcnew String(bstr);
            Console::Write("<{0}>", strName);
            ::SysFreeString(bstr);
            bstr = NULL;

            Marshal::ThrowExceptionForHR(node->get_text(&bstr));
        }
    }
}
```

```

        Console::Write(gcnew String(bstr));
        ::SysFreeString(bstr);
        bstr = NULL;

        Console::WriteLine("</{0}>", strName);
    }
    finally {
        ::SysFreeString(bstr);
    }
}

com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // stream some xml into the document
        doc.LoadXml("<word>persnickety</word>");

        // write the document to the console
        doc.WriteTopLevelNode();
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}

```

```
<#document>persnickety</#document>
```

## ptr::Release

Releases all owned references on the COM object.

```
void Release();
```

### Remarks

Calling this function releases all owned references on the COM object and sets the internal handle to the COM object to `nullptr`. If no other references on the COM object exist, it will be destroyed.

### Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. The `ReplaceDocument` member function uses `Release` to release any prior document object before attaching the new document.

```

// comprtr_release.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an

```

```

// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // replace currently held COM object with another one
    void ReplaceDocument(IXMLDOMDocument* pDoc) {
        // release current document object
        m_ptrDoc.Release();
        // attach the new document object
        m_ptrDoc.Attach(pDoc);
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that creates a raw XML DOM Document object
IXMLDOMDocument* CreateDocument() {
    IXMLDOMDocument* pDoc = NULL;
    Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
        CLSCTX_INPROC_SERVER, IID_IXMLDOMDocument, (void**)&pDoc));
    return pDoc;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // get another document object from unmanaged function and
        // store it in place of the one held by our ref class
        pDoc = CreateDocument();
        doc.ReplaceDocument(pDoc);
        // no further need for raw object reference
        pDoc->Release();
        pDoc = NULL;
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}
}

```

### ptr::operator->

Member access operator, used to call methods on the owned COM object.

```
_detail::smart_com_ptr<_interface_type> operator->();
```

## Return value

A `smart_com_ptr` to the COM object.

## Exceptions

Internally, `QueryInterface` is called on the owned COM object and any error `HRESULT` is converted to an exception by `ThrowExceptionForHR`.

## Remarks

This operator allows you to call methods of the owned COM object. It returns a temporary `smart_com_ptr` that automatically handles its own `AddRef` and `Release`.

## Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. The `WriteDocument` function uses `operator->` to call the `get.firstChild` member of the document object.

```
// comprtr_op_member.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // add a reference to and return the COM object
    // but keep an internal reference to the object
    IXMLDOMDocument* GetDocument() {
        return m_ptrDoc.GetInterface();
    }

    // simplified function that only writes the first node
    void WriteDocument() {
        IXMLDOMNode* pNode = NULL;
        BSTR bstr = NULL;

        try {
            // use operator -> to call XML Doc member
            Marshal::ThrowExceptionForHR(m_ptrDoc->get.firstChild(&pNode));
            if (NULL != pNode) {
                // write out the xml
                Marshal::ThrowExceptionForHR(pNode->get_nodeName(&bstr));
                String^ strName = gcnew String(bstr);
                Console::Write("<{0}>", strName);
                ::SysFreeString(bstr);
                bstr = NULL;

                Marshal::ThrowExceptionForHR(pNode->get_text(&bstr));
                Console::Write(gcnew String(bstr));
                ::SysFreeString(bstr);
                bstr = NULL;

                Console::WriteLine("</{0}>", strName);
            }
        }
    }
}
```

```

        }
    }
    finally {
        if (NULL != pNode) {
            pNode->Release();
        }
        ::SysFreeString(bstr);
    }
}

// note that the destructor will call the com::ptr destructor
// and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that loads XML into a raw XML DOM Document object
HRESULT LoadXml(IXMLDOMDocument* pDoc, BSTR bstrXml) {
    HRESULT hr = S_OK;
    VARIANT_BOOL bSuccess;
    hr = pDoc->loadXML(bstrXml, &bSuccess);
    if (S_OK == hr && !bSuccess) {
        hr = E_FAIL;
    }
    return hr;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;
    BSTR bstrXml = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        bstrXml = ::SysAllocString(L"<word>persnickety</word>");
        if (NULL == bstrXml) {
            throw gcnew OutOfMemoryException("bstrXml");
        }
        // detach the document object from the ref class
        pDoc = doc.GetDocument();
        // use unmanaged function and raw object to load xml
        Marshal::ThrowExceptionForHR(LoadXml(pDoc, bstrXml));
        // release reference to document object (but ref class still references it)
        pDoc->Release();
        pDoc = NULL;

        // call another function on the ref class
        doc.WriteDocument();
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}

```

```
<word>persnickety</word>
```

## ptr::operator=

Attaches a COM object to a `com::ptr`.

```
ptr<_interface_type> % operator=(  
    _interface_type * _right  
)
```

### Parameters

`_right`

The COM interface pointer to attach.

### Return value

A tracking reference on the `com::ptr`.

### Exceptions

If the `com::ptr` already owns a reference to a COM object, `operator=` throws [InvalidOperationException](#).

### Remarks

Assigning a COM object to a `com::ptr` references the COM object but doesn't release the caller's reference to it.

This operator has the same effect as `Attach`.

### Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object.

The `ReplaceDocument` member function first calls `Release` on any previously owned object and then uses

`operator=` to attach a new document object.

```
// comprtr_op_assign.cpp  
// compile with: /clr /link msxml2.lib  
#include <msxml2.h>  
#include <msclr\com\ptr.h>  
  
#import <msxml3.dll> raw_interfaces_only  
  
using namespace System;  
using namespace System::Runtime::InteropServices;  
using namespace msclr;  
  
// a ref class that uses a com::ptr to contain an  
// IXMLDOMDocument object  
ref class XmlDocument {  
public:  
    // construct the internal com::ptr with a null interface  
    // and use CreateInstance to fill it  
    XmlDocument(String^ progid) {  
        m_ptrDoc.CreateInstance(progid);  
    }  
  
    // replace currently held COM object with another one  
    void ReplaceDocument(IXMLDOMDocument* pDoc) {  
        // release current document object  
        m_ptrDoc.Release();  
        // attach the new document object  
        m_ptrDoc = pDoc;  
    }  
  
    // note that the destructor will call the com::ptr destructor  
    // and automatically release the reference to the COM object  
  
private:
```

```

com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that creates a raw XML DOM Document object
IXMLDOMDocument* CreateDocument() {
    IXMLDOMDocument* pDoc = NULL;
    Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
        CLSCTX_INPROC_SERVER, IID_IXMLDOMDocument, (void**)&pDoc));
    return pDoc;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // get another document object from unmanaged function and
        // store it in place of the one held by the ref class
        pDoc = CreateDocument();
        doc.ReplaceDocument(pDoc);
        // no further need for raw object reference
        pDoc->Release();
        pDoc = NULL;
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}

```

## ptr::operator bool

Operator for using `com::ptr` in a conditional expression.

```
operator bool();
```

### Return value

`true` if the owned COM object is valid; `false` otherwise.

### Remarks

The owned COM object is valid if it's not `nullptr`.

This operator converts to `_detail_class::_safe_bool` which is safer than `bool` because it can't be converted to an integral type.

### Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object. The `CreateInstance` member function uses `operator bool` after creating the new document object to determine if it's valid and writes to the console if it is.

```

// comptr_op_bool.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    void CreateInstance(String^ progid) {
        if (!m_ptrDoc) {
            m_ptrDoc.CreateInstance(progid);
            if (m_ptrDoc) { // uses operator bool
                Console::WriteLine("DOM Document created.");
            }
        }
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object
}

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        XmlDocument doc;
        // create the instance from a progid string
        doc.CreateInstance("Msxml2.DOMDocument.3.0");
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}

```

DOM Document created.

## ptr::operator!

Operator to determine if the owned COM object is invalid.

```
bool operator!();
```

### Return value

`true` if the owned COM object is invalid; `false` otherwise.

### Remarks

The owned COM object is valid if it's not `nullptr`.

### Example

This example implements a CLR class that uses a `com::ptr` to wrap its private member `IXMLDOMDocument` object.

The `GetInstance` member function uses `operator!` to determine if a document object is already owned, and only creates a new instance if the object is invalid.

```
// comptr_op_not.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    void CreateInstance(String^ progid) {
        if (!m_ptrDoc) {
            m_ptrDoc.CreateInstance(progid);
            if (m_ptrDoc)
                Console::WriteLine("DOM Document created.");
        }
    }
};

// note that the destructor will call the com::ptr destructor
// and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        XmlDocument doc;
        // create the instance from a progid string
        doc.CreateInstance("Msxml2.DOMDocument.3.0");
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}
```

DOM Document created.

# Exceptions in C++/CLI

9/20/2022 • 2 minutes to read • [Edit Online](#)

The articles in this section of the documentation explain exception handling in C++/CLI and how it differs from standard exception handling.

## Related Articles

TITLE	DESCRIPTION
<a href="#">Basic Concepts in Using Managed Exceptions</a>	Discusses exception handling in managed applications.
<a href="#">Differences in Exception Handling Behavior Under /CLR</a>	Discusses the differences between standard exception handling and exception handling in C++/CLI.
<a href="#">finally</a>	Describes the <code>finally</code> block that's used to clean up resources that are left after an exception occurs.
<a href="#">How to: Catch Exceptions in Native Code Thrown from MSIL</a>	Demonstrates how to use <code>__try</code> and <code>__except</code> to catch exceptions in native code that are thrown from MSIL.
<a href="#">How to: Define and Install a Global Exception Handler</a>	Demonstrates how to capture unhandled exceptions.
<a href="#">.NET Programming with C++/CLI (Visual C++)</a>	The top-level article for .NET programming in the Visual C++ documentation.

# Basic Concepts in Using Managed Exceptions

9/20/2022 • 3 minutes to read • [Edit Online](#)

This topic discusses exception handling in managed applications. That is, an application that is compiled with the `/clr` compiler option.

## In this topic

- [Throwing Exceptions Under /clr](#)
- [Try/Catch Blocks for CLR Extensions](#)

## Remarks

If you compile with the `/clr` option, you can handle CLR exceptions as well as standard [Exception](#) class provides many useful methods for processing CLR exceptions and is recommended as a base class for user-defined exception classes.

Catching exception types derived from an interface is not supported under `/clr`. Also, the common language runtime does not permit you to catch stack overflow exceptions; a stack overflow exception will terminate the process.

For more information about differences in exception handling in managed and unmanaged applications, see [Differences in Exception Handling Behavior Under Managed Extensions for C++](#).

## Throwing Exceptions Under `/clr`

The C++ throw expression is extended to throw a handle to a CLR type. The following example creates a custom exception type and then throws an instance of that type:

```
// clr_exception_handling.cpp
// compile with: /clr /c
ref struct MyStruct: public System::Exception {
public:
    int i;
};

void GlobalFunction() {
    MyStruct^ pMyStruct = gcnew MyStruct;
    throw pMyStruct;
}
```

A value type must be boxed before being thrown:

```

// clr_exception_handling_2.cpp
// compile with: /clr /c
value struct MyValueStruct {
    int i;
};

void GlobalFunction() {
    MyValueStruct v = {11};
    throw (MyValueStruct ^)v;
}

```

## Try/Catch Blocks for CLR Extensions

The same `try / catch` block structure can be used for catching both CLR and native exceptions:

```

// clr_exception_handling_3.cpp
// compile with: /clr
using namespace System;
ref struct MyStruct : public Exception {
public:
    int i;
};

struct CMyClass {
public:
    double d;
};

void GlobalFunction() {
    MyStruct^ pMyStruct = gcnew MyStruct;
    pMyStruct->i = 11;
    throw pMyStruct;
}

void GlobalFunction2() {
    CMyClass c = {2.0};
    throw c;
}

int main() {
    for ( int i = 1; i >= 0; --i ) {
        try {
            if ( i == 1 )
                GlobalFunction2();
            if ( i == 0 )
                GlobalFunction();
        }
        catch ( CMyClass& catchC ) {
            Console::WriteLine( "In 'catch(CMyClass& catchC)'" );
            Console::WriteLine( catchC.d );
        }
        catch ( MyStruct^ catchException ) {
            Console::WriteLine( "In 'catch(MyStruct^ catchException)'" );
            Console::WriteLine( catchException->i );
        }
    }
}

```

## Output

```
In 'catch(CMyClass& catchC)'
2
In 'catch(MyStruct^ catchException)'
11
```

## Order of Unwinding for C++ Objects

Unwinding occurs for any C++ objects with destructors that may be on the run-time stack between the throwing function and the handling function. Because CLR types are allocated on the heap, unwinding does not apply to them.

The order of events for a thrown exception is as follows:

1. The runtime walks the stack looking for the appropriate catch clause, or in the case of SEH, an except filter for SEH, to catch the exception. Catch clauses are searched first in lexical order, and then dynamically down the call stack.
2. Once the correct handler is found, the stack is unwound to that point. For each function call on the stack, its local objects are destructed and `_finally` blocks are executed, from most nested outward.
3. Once the stack is unwound, the catch clause is executed.

## Catching Unmanaged Types

When an unmanaged object type is thrown, it is wrapped with an exception of type `SEHException`. When searching for the appropriate `catch` clause, there are two possibilities.

- If a native C++ type is encountered, the exception is unwrapped and compared to the type encountered. This comparison allows a native C++ type to be caught in the normal way.
- However, if a `catch` clause of type `SEHException` or any of its base classes is examined first, the clause will intercept the exception. Therefore, you should place all catch clauses that catch native C++ types first before any catch clauses of CLR types.

Note that

```
catch(Object^)
```

and

```
catch(...)
```

will both catch any thrown type including SEH exceptions.

If an unmanaged type is caught by `catch(Object^)`, it will not destroy the thrown object.

When throwing or catching unmanaged exceptions, we recommend that you use the `/EHsc` compiler option instead of `/EHs` or `/EHa`.

## See also

[Exception Handling](#)

[safe\\_cast](#)

[Exception Handling](#)

# Differences in Exception Handling Behavior Under /CLR

9/20/2022 • 4 minutes to read • [Edit Online](#)

[Basic Concepts in Using Managed Exceptions](#) discusses exception handling in managed applications. In this topic, differences from the standard behavior of exception handling and some restrictions are discussed in detail. For more information, see [The \\_set\\_se\\_translator Function](#).

## Jumping Out of a Finally Block

In native C/C++ code, jumping out of a `_finally` block using structured exception handling (SEH) is allowed although it produces a warning. Under `/clr`, jumping out of a `finally` block causes an error:

```
// clr_exception_handling_4.cpp
// compile with: /clr
int main() {
    try {}
    finally {
        return 0;    // also fails with goto, break, continue
    }
} // C3276
```

## Raising Exceptions Within an Exception Filter

When an exception is raised during the processing of an [exception filter](#) within managed code, the exception is caught and treated as if the filter returns 0.

This is in contrast to the behavior in native code where a nested exception is raised, the `ExceptionRecord` field in the `EXCEPTION_RECORD` structure (as returned by [GetExceptionInformation](#)) is set, and the `ExceptionFlags` field sets the 0x10 bit. The following example illustrates this difference in behavior:

```

// clr_exception_handling_5.cpp
#include <windows.h>
#include <stdio.h>
#include <assert.h>

#ifndef false
#define false 0
#endif

int *p;

int filter(PEXCEPTION_POINTERS ExceptionPointers) {
    PEXCEPTION_RECORD ExceptionRecord =
        ExceptionPointers->ExceptionRecord;

    if (((ExceptionRecord->ExceptionFlags & 0x10) == 0) {
        // not a nested exception, throw one
        *p = 0; // throw another AV
    }
    else {
        printf("Caught a nested exception\n");
        return 1;
    }

    assert(false);

    return 0;
}

void f(void) {
    __try {
        *p = 0; // throw an AV
    }
    __except(filter(GetExceptionInformation())) {
        printf_s("We should execute this handler if "
                "compiled to native\n");
    }
}

int main() {
    __try {
        f();
    }
    __except(1) {
        printf_s("The handler in main caught the "
                "exception\n");
    }
}

```

## Output

```

Caught a nested exception
We should execute this handler if compiled to native

```

## Disassociated Rethrows

/clr does not support rethrowing an exception outside of a catch handler (known as a disassociated rethrow). Exceptions of this type are treated as a standard C++ rethrow. If a disassociated rethrow is encountered when there is an active managed exception, the exception is wrapped as a C++ exception and then rethrown. Exceptions of this type can only be caught as an exception of type [SEHException](#).

The following example demonstrates a managed exception rethrown as a C++ exception:

```

// clr_exception_handling_6.cpp
// compile with: /clr
using namespace System;
#include <cassert.h>
#include <stdio.h>

void rethrow( void ) {
    // This rethrow is a disassociated rethrow.
    // The exception would be masked as SEHException.
    throw;
}

int main() {
    try {
        try {
            throw gcnew ApplicationException;
        }
        catch ( ApplicationException^ ) {
            rethrow();
            // If the call to rethrow() is replaced with
            // a throw statement within the catch handler,
            // the rethrow would be a managed rethrow and
            // the exception type would remain
            // System::ApplicationException
        }
    }
    catch ( ApplicationException^ ) {
        assert( false );

        // This will not be executed since the exception
        // will be masked as SEHException.
    }
    catch ( Runtime::InteropServices::SEHException^ ) {
        printf_s("caught an SEH Exception\n" );
    }
}

```

## Output

```
caught an SEH Exception
```

## Exception Filters and EXCEPTION\_CONTINUE\_EXECUTION

If a filter returns `EXCEPTION_CONTINUE_EXECUTION` in a managed application, it is treated as if the filter returned `EXCEPTION_CONTINUE_SEARCH`. For more information on these constants, see [try-except Statement](#).

The following example demonstrates this difference:

```

// clr_exception_handling_7.cpp
#include <windows.h>
#include <stdio.h>
#include <assert.h>

int main() {
    int Counter = 0;
    __try {
        __try {
            Counter -= 1;
            RaiseException (0xe0000000|'seh',
                            0, 0, 0);
            Counter -= 2;
        }
        __except (Counter) {
            // Counter is negative,
            // indicating "CONTINUE EXECUTE"
            Counter -= 1;
        }
    }
    __except(1) {
        Counter -= 100;
    }

    printf_s("Counter=%d\n", Counter);
}

```

## Output

```
Counter=-3
```

## The \_set\_se\_translator Function

The translator function, set by a call to `_set_se_translator`, affects only catches in unmanaged code. The following example demonstrates this limitation:

```

// clr_exception_handling_8.cpp
// compile with: /clr /EHs
#include <iostream>
#include <windows.h>
#include <eh.h>
#pragma warning (disable: 4101)
using namespace std;
using namespace System;

#define MYEXCEPTION_CODE 0xe0000101

class CMyException {
public:
    unsigned int m_ErrorCode;
    EXCEPTION_POINTERS * m_pExp;

    CMyException() : m_ErrorCode( 0 ), m_pExp( NULL ) {}

    CMyException( unsigned int i, EXCEPTION_POINTERS * pExp )
        : m_ErrorCode( i ), m_pExp( pExp ) {}

    CMyException( CMyException& c ) : m_ErrorCode( c.m_ErrorCode ),
                                    m_pExp( c.m_pExp ) {}

    friend ostream& operator <<
        ( ostream& out, const CMyException& inst ) {
        return out << "CMyException[\n" <<

```

```

        "Error Code: " << inst.m_ErrorCode << "]";
    }

};

#pragma unmanaged
void my_trans_func( unsigned int u, PEXCEPTION_POINTERS pExp ) {
    cout << "In my_trans_func.\n";
    throw CMyException( u, pExp );
}

#pragma managed
void managed_func() {
    try {
        RaiseException( MYEXCEPTION_CODE, 0, 0, 0 );
    }
    catch ( CMyException x ) {}
    catch ( ... ) {
        printf_s("This is invoked since "
                "_set_se_translator is not "
                "supported when /clr is used\n");
    }
}

#pragma unmanaged
void unmanaged_func() {
    try {
        RaiseException( MYEXCEPTION_CODE,
                        0, 0, 0 );
    }
    catch ( CMyException x ) {
        printf("Caught an SEH exception with "
               "exception code: %x\n", x.m_ErrorCode );
    }
    catch ( ... ) {}
}

// #pragma managed
int main( int argc, char ** argv ) {
    _set_se_translator( my_trans_func );

    // It does not matter whether the translator function
    // is registered in managed or unmanaged code
    managed_func();
    unmanaged_func();
}

```

## Output

```

This is invoked since _set_se_translator is not supported when /clr is used
In my_trans_func.
Caught an SEH exception with exception code: e0000101

```

## See also

[Exception Handling](#)

[safe\\_cast](#)

[Exception Handling in MSVC](#)

# finally

9/20/2022 • 2 minutes to read • [Edit Online](#)

In addition to `try` and `catch` clauses, CLR exception handling supports a `finally` clause. The semantics are identical to the `__finally` block in structured exception handling (SEH). A `__finally` block can follow a `try` or `catch` block.

## Remarks

The purpose of the `finally` block is to clean up any resources left after the exception occurred. Note that the `finally` block is always executed, even if no exception was thrown. The `catch` block is only executed if a managed exception is thrown within the associated `try` block.

`finally` is a context-sensitive keyword; see [Context-Sensitive Keywords](#) for more information.

## Example

The following example demonstrates a simple `finally` block:

```
// keyword__finally.cpp
// compile with: /clr
using namespace System;

ref class MyException: public System::Exception{};

void ThrowMyException() {
    throw gcnew MyException;
}

int main() {
    try {
        ThrowMyException();
    }
    catch ( MyException^ e ) {
        Console::WriteLine( "in catch" );
        Console::WriteLine( e->GetType() );
    }
    finally {
        Console::WriteLine( "in finally" );
    }
}
```

```
in catch
MyException
in finally
```

## See also

[Exception Handling](#)

# How to: Catch exceptions in native code thrown from MSIL

9/20/2022 • 2 minutes to read • [Edit Online](#)

In native code, you can catch native C++ exception from MSIL. You can catch CLR exceptions with `__try` and `__except`.

For more information, see [Structured Exception Handling \(C/C++\)](#) and [Modern C++ best practices for exceptions and error handling](#).

## Example 1

The following sample defines a module with two functions, one that throws a native exception, and another that throws an MSIL exception.

```
// catch_MSIL_in_native.cpp
// compile with: /clr /c
void Test() {
    throw ("error");
}

void Test2() {
    throw (gcnew System::Exception("error2"));
}
```

## Example 2

The following sample defines a module that catches a native and MSIL exception.

```
// catch_MSIL_in_native_2.cpp
// compile with: /clr catch_MSIL_in_native.obj
#include <iostream>
using namespace std;
void Test();
void Test2();

void Func() {
    // catch any exception from MSIL
    // should not catch Visual C++ exceptions like this
    // runtime may not destroy the object thrown
    __try {
        Test2();
    }
    __except(1) {
        cout << "caught an exception" << endl;
    }
}

int main() {
    // catch native C++ exception from MSIL
    try {
        Test();
    }
    catch(char * s) {
        cout << s << endl;
    }
    Func();
}
```

```
error
caught an exception
```

## See also

[Exception handling](#)

# How to: Define and Install a Global Exception Handler

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following code example demonstrates how unhandled exceptions can be captured. The example form contains a button that, when pressed, performs a null reference, causing an exception to be thrown. This functionality represents a typical code failure. The resulting exception is caught by the application-wide exception handler installed by the main function.

This is accomplished by binding a delegate to the [ThreadException](#) event. In this case, subsequent exceptions are then sent to the `App::OnUnhandled` method.

## Example

```

// global_exception_handler.cpp
// compile with: /clr
#using <system.dll>
#using <system.drawing.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Threading;
using namespace System::Drawing;
using namespace System::Windows::Forms;

ref class MyForm : public Form
{
    Button^ b;
public:
    MyForm( )
    {
        b = gcnew Button( );
        b->Text = "Do Null Access";
        b->Size = Drawing::Size(150, 30);
        b->Click += gcnew EventHandler(this, &MyForm::OnClick);
        Controls->Add(b);
    }
    void OnClick(Object^ sender, EventArgs^ args)
    {
        // do something illegal, like call through a null pointer...
        Object^ o = nullptr;
        o->ToString( );
    }
};

ref class App
{
public:
    static void OnUnhandled(Object^ sender, ThreadExceptionEventArgs^ e)
    {
        MessageBox::Show(e->Exception->Message, "Global Exception");
        Application::ExitThread( );
    }
};

int main()
{
    Application::ThreadException += gcnew
        ThreadExceptionEventHandler(App::OnUnhandled);

    MyForm^ form = gcnew MyForm( );
    Application::Run(form);
}

```

## See also

[Exception Handling](#)

# Boxing (C++/CLI)

9/20/2022 • 2 minutes to read • [Edit Online](#)

Boxing is the process of converting a value type to the type `object` or to any interface type that's implemented by the value type. When the common language runtime (CLR) boxes a value type, it wraps the value in a `System.Object` and stores it on the managed heap. Unboxing extracts the value type from the object. Boxing is implicit; unboxing is explicit.

## Related Articles

TITLE	DESCRIPTION
<a href="#">How to: Explicitly Request Boxing</a>	Describes how to explicitly request boxing on a variable.
<a href="#">How to: Use gcnew to Create Value Types and Use Implicit Boxing</a>	Shows how to use <code>gcnew</code> to create a boxed value type that can be placed on the managed, garbage-collected heap.
<a href="#">How to: Unbox</a>	Shows how to unbox and modify a value.
<a href="#">Standard Conversions and Implicit Boxing</a>	Shows that a standard conversion is chosen by the compiler over a conversion that requires boxing.
<a href="#">.NET Programming with C++/CLI (Visual C++)</a>	The top-level article for .NET programming in the Visual C++ documentation.

# How to: Explicitly Request Boxing

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can explicitly request boxing by assigning a variable to a variable of type `Object`.

## Example

```
// vc++v2_explicit_boxing3.cpp
// compile with: /clr
using namespace System;

void f(int i) {
    Console::WriteLine("f(int i)");
}

void f(Object ^o) {
    Console::WriteLine("f(Object^ o)");
}

int main() {
    int i = 5;
    Object ^ O = i;    // forces i to be boxed
    f(i);
    f(O);
    f( (Object^)i ); // boxes i
}
```

```
f(int i)
f(Object^ o)
f(Object^ o)
```

## See also

[Boxing](#)

# How to: Use gcnew to Create Value Types and Use Implicit Boxing

9/20/2022 • 2 minutes to read • [Edit Online](#)

Using [gcnew](#) on a value type will create a boxed value type, which can then be placed on the managed, garbage-collected heap.

## Example

```
// vcmcppv2_explicit_boxing4.cpp
// compile with: /clr
public value class V {
public:
    int m_i;
    V(int i) : m_i(i) {}
};

public ref struct TC {
    void do_test(V^ v) {
        if (v != nullptr)
            ;
        else
            ;
    }
};

int main() {
    V^ v = gcnew V(42);
    TC^ tc = gcnew TC;
    tc->do_test(v);
}
```

## See also

[Boxing](#)

# How to: Unbox

9/20/2022 • 2 minutes to read • [Edit Online](#)

Shows how to unbox and modify a value.

## Example

```
// vcmcppv2_unboxing.cpp
// compile with: /clr
using namespace System;

int main() {
    int ^ i = gcnew int(13);
    int j;
    Console::WriteLine(*i);    // unboxing
    *i = 14;    // unboxing and assignment
    Console::WriteLine(*i);
    j = safe_cast<int>(i);    // unboxing and assignment
    Console::WriteLine(j);
}
```

```
13
14
14
```

## See also

[Boxing](#)

# Standard Conversions and Implicit Boxing

9/20/2022 • 2 minutes to read • [Edit Online](#)

A standard conversion will be chosen by the compiler over a conversion that requires boxing.

## Example

```
// clr_implicit_boxing_Std_conversion.cpp
// compile with: /clr
int f3(int ^ i) {    // requires boxing
    return 1;
}

int f3(char c) {    // no boxing required, standard conversion
    return 2;
}

int main() {
    int i = 5;
    System::Console::WriteLine(f3(i));
}
```

2

## See also

[Boxing](#)