`/clr` (Common Language Runtime Compilation)

Article • 10/29/2021

Enables applications and components to use features from the common language runtime (CLR) and enables C++/CLI compilation.

Syntax

> `/clr`[`:`*options*]

Arguments

options
One or more of the following comma-separated arguments.

- none

  With no options, `/clr` creates metadata for the component. The metadata can be consumed by other CLR applications, and enables the component to consume types and data in the metadata of other CLR components. For more information, see Mixed (Native and Managed) Assemblies.

- `netcore`

  Available starting in Visual Studio 2019 version 16.4, `/clr:netcore` creates metadata and code for the component using the latest cross-platform .NET framework, also known as .NET Core. The metadata can be consumed by other .NET Core applications. And, the option enables the component to consume types and data in the metadata of other .NET Core components.

- `nostdlib`

  Instructs the compiler to ignore the default `\clr` directory. The compiler produces errors if you include multiple versions of a DLL, such as System.dll. This option lets you specify the specific framework to use during compilation.

- `pure`

`/clr:pure` **is deprecated**. The option is removed in Visual Studio 2017 and later. We recommend that you port code that must be pure MSIL to C#.

- `safe`

  `/clr:safe` **is deprecated**. The option is removed in Visual Studio 2017 and later. We recommend that you port code that must be safe MSIL to C#.

- `noAssembly`

  `/clr:noAssembly` **is deprecated**. Use /LN (Create MSIL Module) instead.

  Tells the compiler not to insert an assembly manifest into the output file. By default, the `noAssembly` option isn't in effect.

  A managed program that doesn't have assembly metadata in the manifest is known as a *module*. The `noAssembly` option can be used only to produce a module. If you compile by using /c and `/clr:noAssembly`, then specify the /NOASSEMBLY option in the linker phase to create a module.

  Before Visual Studio 2005, `/clr:noAssembly` required `/LD`. `/LD` is now implied when you specify `/clr:noAssembly`.

- `initialAppDomain`

  `initialAppDomain` **is obsolete**. Enables a C++/CLI application to run on version 1 of the CLR. An application that's compiled by using `initialAppDomain` shouldn't be used by an application that uses ASP.NET because it's not supported in version 1 of the CLR.

Remarks

Managed code is code that can be inspected and managed by the CLR. Managed code can access managed objects. For more information, see /clr Restrictions.

For information about how to develop applications that define and consume managed types in C++, see Component Extensions for Runtime Platforms.

An application compiled by using `/clr` may or may not contain managed data.

To enable debugging on a managed application, see /ASSEMBLYDEBUG (Add DebuggableAttribute).

Only CLR types are instantiated on the garbage-collected heap. For more information, see Classes and Structs. To compile a function to native code, use the `unmanaged` pragma. For more information, see managed, unmanaged.

By default, `/clr` isn't in effect. When `/clr` is in effect, `/MD` is also in effect. For more information, see /MD, /MT, /LD (Use Run-Time Library). `/MD` ensures that the dynamically linked, multithreaded versions of the runtime routines are selected from the standard header files. Multithreading is required for managed programming because the CLR garbage collector runs finalizers in an auxiliary thread.

If you compile by using `/c`, you can specify the CLR type of the resulting output file by using the /CLRIMAGETYPE linker option.

`/clr` implies `/EHa`, and no other `/EH` options are supported for `/clr`. For more information, see /EH (Exception Handling Model).

For information about how to determine the CLR image type of a file, see /CLRHEADER.

All modules passed to a given invocation of the linker must be compiled by using the same run-time library compiler option (`/MD` or `/LD`).

Use the /ASSEMBLYRESOURCE linker option to embed a resource in an assembly. /DELAYSIGN, /KEYCONTAINER, and /KEYFILE linker options also let you customize how an assembly is created.

When `/clr` is used, the `_MANAGED` symbol is defined to be 1. For more information, see Predefined macros.

The global variables in a native object file are initialized first (during `DllMain` if the executable is a DLL), and then the global variables in the managed section are initialized (before any managed code is run). #pragma init_seg only affects the order of initialization in the managed and unmanaged categories.

Metadata and Unnamed Classes

Unnamed classes appear in metadata under names such as `$UnnamedClass$<crc-of-current-file-name>$<index>$`, where `<index>` is a sequential count of the unnamed classes in the compilation. For example, the following code sample generates an unnamed class in metadata.

C++

```
// clr_unnamed_class.cpp
// compile by using: /clr /LD
```

```
class {} x;
```

Use ildasm.exe to view metadata.

To set this compiler option in the Visual Studio development environment

1. Open the project's **Property Pages** dialog box. For details, see Set C++ compiler and build properties in Visual Studio.
2. Set the **Configuration** dropdown to **All configurations**, and set the **Platform** dropdown to **All Platforms**.
3. Select the **Configuration Properties** > **C/C++** > **General** page.
4. Modify the **Common Language Runtime Support** property. Choose **OK** to save your changes.

> Note
>
> In the Visual Studio IDE, the `/clr` compiler option can be individually set on the **Configuration Properties** > **C/C++** > **General** page of the Property Pages dialog. However, we recommend you use a CLR template to create your project. It sets all of the properties required for successful creation of a CLR component. Another way to set these properties is to use the **Common Language Runtime Support** property on the **Configuration Properties** > **Advanced** page of the Property Pages dialog. This property sets all the other CLR-related tool options at once.

To set this compiler option programmatically

- See CompileAsManaged.

See also

MSVC Compiler Options
MSVC Compiler Command-Line Syntax

Feedback