Lifetime, Scope, Visibility, and Linkage

**Visual Studio 2013**

[Other Versions](javascript:;)

To understand how a C program works, you must understand the rules that determine how variables and functions can be used in the program. Several concepts are crucial to understanding these rules:

* [Lifetime](https://msdn.microsoft.com/en-us/library/30k8f0kc(v=vs.120).aspx)
* [Scope and visibility](https://msdn.microsoft.com/en-us/library/awt60xs1(v=vs.120).aspx)
* [Linkage](https://msdn.microsoft.com/en-us/library/teta4z44(v=vs.120).aspx)

Lifetime

**Visual Studio 2013**

[Other Versions](javascript:;)

"Lifetime" is the period during execution of a program in which a variable or function exists. The storage duration of the identifier determines its lifetime.

An identifier declared with the *storage-class-specifier* **static** has static storage duration. Identifiers with static storage duration (also called "global") have storage and a defined value for the duration of a program. Storage is reserved and the identifier's stored value is initialized only once, before program startup. An identifier declared with external or internal linkage also has static storage duration (see [Linkage](https://msdn.microsoft.com/en-us/library/teta4z44(v=vs.120).aspx)).

An identifier declared without the **static** storage-class specifier has automatic storage duration if it is declared inside a function. An identifier with automatic storage duration (a "local identifier") has storage and a defined value only within the block where the identifier is defined or declared. An automatic identifier is allocated new storage each time the program enters that block, and it loses its storage (and its value) when the program exits the block. Identifiers declared in a function with no linkage also have automatic storage duration.

The following rules specify whether an identifier has global (static) or local (automatic) lifetime:

* All functions have static lifetime. Therefore they exist at all times during program execution. Identifiers declared at the external level (that is, outside all blocks in the program at the same level of function definitions) always have global (static) lifetimes.
* If a local variable has an initializer, the variable is initialized each time it is created (unless it is declared as **static**). Function parameters also have local lifetime. You can specify global lifetime for an identifier within a block by including the **static** storage-class specifier in its declaration. Once declared**static**, the variable retains its value from one entry of the block to the next.

Although an identifier with a global lifetime exists throughout the execution of the source program (for example, an externally declared variable or a local variable declared with the **static** keyword), it may not be visible in all parts of the program. See [Scope and Visibility](https://msdn.microsoft.com/en-us/library/awt60xs1(v=vs.120).aspx) for information about visibility, and see[Storage Classes](https://msdn.microsoft.com/en-us/library/w9hwbe3d(v=vs.120).aspx) for a discussion of the *storage-class-specifier* nonterminal.

Memory can be allocated as needed (dynamic) if created through the use of special library routines such as **malloc**. Since dynamic memory allocation uses library routines, it is not considered part of the language. See the [malloc](https://msdn.microsoft.com/en-us/library/6ewkz86d(v=vs.120).aspx) function in the *Run-Time Library Reference*.

# Scope and Visibility

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

An identifier's "visibility" determines the portions of the program in which it can be referenced — its "scope." An identifier is visible (i.e., can be used) only in portions of a program encompassed by its "scope," which may be limited (in order of increasing restrictiveness) to the file, function, block, or function prototype in which it appears. The scope of an identifier is the part of the program in which the name can be used. This is sometimes called "lexical scope." There are four kinds of scope: function, file, block, and function prototype.

All identifiers except labels have their scope determined by the level at which the declaration occurs. The following rules for each kind of scope govern the visibility of identifiers within a program:

File scope

The declarator or type specifier for an identifier with file scope appears outside any block or list of parameters and is accessible from any place in the translation unit after its declaration. Identifier names with file scope are often called "global" or "external." The scope of a global identifier begins at the point of its definition or declaration and terminates at the end of the translation unit.

Function scope

A label is the only kind of identifier that has function scope. A label is declared implicitly by its use in a statement. Label names must be unique within a function. (For more information about labels and label names, see [The goto and Labeled Statements](https://msdn.microsoft.com/en-us/library/2c002fdz(v=vs.120).aspx).)

Block scope

The declarator or type specifier for an identifier with block scope appears inside a block or within the list of formal parameter declarations in a function definition. It is visible only from the point of its declaration or definition to the end of the block containing its declaration or definition. Its scope is limited to that block and to any blocks nested in that block and ends at the curly brace that closes the associated block. Such identifiers are sometimes called "local variables."

Function-prototype scope

The declarator or type specifier for an identifier with function-prototype scope appears within the list of parameter declarations in a function prototype (not part of the function declaration). Its scope terminates at the end of the function declarator.

The appropriate declarations for making variables visible in other source files are described in [Storage Classes](https://msdn.microsoft.com/en-us/library/w9hwbe3d(v=vs.120).aspx). However, variables and functions declared at the external level with the **static** storage-class specifier are visible only within the source file in which they are defined. All other functions are globally visible.

# Linkage

**Visual Studio 2013**

[Other Versions](javascript:;)

Identifier names can refer to different identifiers in different scopes. An identifier declared in different scopes or in the same scope more than once can be made to refer to the same identifier or function by a process called "linkage." Linkage determines the portions of the program in which an identifier can be referenced (its "visibility"). There are three kinds of linkage: [internal](https://msdn.microsoft.com/en-us/library/cftw3t5e(v=vs.120).aspx), [external](https://msdn.microsoft.com/en-us/library/k8w8btzz(v=vs.120).aspx), and [no linkage](https://msdn.microsoft.com/en-us/library/aa379a5t(v=vs.120).aspx).

# Internal Linkage

**Visual Studio 2013**

[Other Versions](javascript:;)

If the declaration of a file-scope identifier for an object or a function contains the storage-class-specifier **static**, the identifier has internal linkage. Otherwise, the identifier has external linkage. See [Storage Classes](https://msdn.microsoft.com/en-us/library/w9hwbe3d(v=vs.120).aspx) for a discussion of the storage-class-specifier nonterminal.

Within one translation unit, each instance of an identifier with internal linkage denotes the same identifier or function. Internally linked identifiers are unique to a translation unit.

# External Linkage

**Visual Studio 2013**

[Other Versions](javascript:;)

If the first declaration at file-scope level for an identifier does not use the **static** storage-class specifier, the object has external linkage.

If the declaration of an identifier for a function has no storage-class-specifier, its linkage is determined exactly as if it were declared with the storage-class-specifier **extern**. If the declaration of an identifier for an object has file scope and no storage-class-specifier, its linkage is external.

An identifier's name with external linkage designates the same function or data object as does any other declaration for the same name with external linkage. The two declarations can be in the same translation unit or in different translation units. If the object or function also has global lifetime, the object or function is shared by the entire program.

No Linkage

**Visual Studio 2013**

[Other Versions](javascript:;)

If a declaration for an identifier within a block does not include the **extern** storage-class specifier, the identifier has no linkage and is unique to the function.

The following identifiers have no linkage:

* An identifier declared to be anything other than an object or a function
* An identifier declared to be a function parameter
* A block-scope identifier for an object declared without the **extern** storage-class specifier

If an identifier has no linkage, declaring the same name again (in a declarator or type specifier) in the same scope level generates a symbol redefinition error.