.NET Framework unmanaged API reference

07/29/2025

This section includes information on unmanaged APIs that can be used by managed-code-related applications, such as runtime hosts, compilers, disassemblers, obfuscators, debuggers, and profilers.

For unmanaged APIs that can be used with both .NET Framework and .NET, see .NET unmanaged API reference.

In this section

Common Data Types
Lists the common data types that are used, particularly in the unmanaged profiling and debugging APIs.

ALink
Describes the ALink API, which supports the creation of .NET Framework assemblies and unbound modules.

Authenticode
Supports the Authenticode XrML license creation and verification module.

Constants
Describes the constants that are defined in CorSym.idl.

Debugging
Describes the debugging API, which enables a debugger to debug code that runs in the common language runtime (CLR) environment.

Diagnostics Symbol Store
Describes the diagnostics symbol store API, which enables a compiler to generate symbol information for use by a debugger.

Fusion
Describes the fusion API, which enables a runtime host to access the properties of an application's resources in order to locate the correct versions of those resources for the application.

Hosting
Describes the hosting API, which enables unmanaged hosts to integrate the CLR into their applications.

Metadata

Describes the metadata API, which enables a client such as a compiler to generate or access a component's metadata without the types being loaded by the CLR.

Profiling

Describes the profiling API, which enables a profiler to monitor a program's execution by the CLR.

Strong Naming

Describes the strong naming API, which enables a client to administer strong name signing for assemblies.

WMI and Performance Counters

Describes the APIs that wrap calls to Windows Management Instrumentation (WMI) libraries.

Tlbexp Helper Functions

Describes the two helper functions and interface used by the Type Library Exporter (Tlbexp.exe) during the assembly-to-type-library conversion process.

Common Data Types (Unmanaged API Reference)

Article • 09/15/2021

This topic lists simple data types used by the unmanaged APIs for the .NET Framework that are defined by C/C++ `typedef` statements. These data types are typically aliases for C/C++ primitive data types. Typically, the values of these data types are opaque; that is, they are returned by a particular function or method so that they can be passed to other functions or methods without modification.

Expand table

Data type	Definition	Defined in	Description
AppDomainID	`typedef UINT_PTR AppDomainID;`	corprof.h	The identifier of an application domain.
AssemblyID	`typedef UINT_PTR AssemblyID;`	corprof.h	The identifier of an assembly.
ClassID	`typedef UINT_PTR ClassID;`	corprof.h	The identifier of a managed class.
CLRDATA_ADDRESS	`typedef ULONG64 CLRDATA_ADDRESS;`	clrdata.h	A 64-bit memory address.
CLRDATA_ENUM	`typedef ULONG64 CLRDATA_ADDRESS;`	Not Available	A 64-bit memory address.
CONNID	`typedef DWORD CONNID;`	cordebug.h, mscoree.h	The connection identifier for a thread that is connected to an instance of Microsoft SQL Server.
ContextID	`typedef UINT_PTR ContextID;`	corprof.h	The identifier of the context associated with a particular managed thread.
COR_PRF_ELT_INFO	`typedef UINT_PTR COR_PRF_ELT_INFO;`	corprof.h	An opaque handle that represents information about a particular stack frame.

Data type	Definition	Defined in	Description
COR_PRF_FRAME_INFO	`typedef UINT_PTR COR_PRF_FRAME_INFO;`	corprof.h	An opaque handle that points to a stack frame. It is valid only during the callback to which it is passed.
CORDB_ADDRESS	`typedef ULONG64 CORDB_ADDRESS;`	cordebug.h	An address in memory.
CORDB_CONTINUE_STATUS	`typedef DWORD CORDB_CONTINUE_STATUS;`	cordebug.h	The continuation status.
CORDB_REGISTER	`typedef ULONG64 CORDB_REGISTER;`	cordebug.h	The value of a CPU register.
FunctionID	`typedef UINT_PTR FunctionID;`	corprof.h	The identifier of a function or method.
GCHandleID	`typedef UINT_PTR GCHandleID;`	corprof.h	A garbage collection handle.
mdMethodDef	`typedef mdToken mdMethodDef;`	cordebug.h	A method definition token.
mdToken	`typedef UINT32 mdToken;`	corprof.h	A metadata token (a row in a metadata table).
ModuleID	`typedef UINT_PTR ModuleID;`	corprof.h	The identifier of an assembly module.
ObjectID	`typedef UINT_PTR ObjectID;`	corprof.h	The identifier of an object.
PCCOR_SIGNATURE	`typedef SIZE_T PCCOR_SIGNATURE;`	cordebug.h	A pointer to a member or metadata signature.
ProcessID	`typedef UINT_PTR ProcessID;`	corprof.h	The identifier of a managed process.
ReJITID	`typedef UINT_PTR ReJITID;`	corprof.h	The identifier of a jitted function.
SIZE_T	`typedef ULONG_PTR SIZE_T;`	corsym.h	A pointer to a 64-bit memory address.
TASKID	`typedef UINT64 TASKID;`	cordebug.h, mscoree.h	The identifier of an ICLRTask instance.

Data type	Definition	Defined in	Description
ThreadID	`typedef UINT_PTR ThreadID;`	corprof.h	The identifier of a managed thread.

See also

- Unmanaged API Reference

ALink API (Unmanaged API Reference)

Article • 09/15/2021

Supports creating .NET Framework assemblies and unbound modules.

In This Section

AssemblyAttributesGoHere

AssemblyAttributesGoHereM

AssemblyAttributesGoHereS

AssemblyAttributesGoHereSM

AssemblyOptions Enumeration

CreateALink Function

GetALinkMessageDll Function

IALink Interface

IALink2 Interface

IALink3 Interface

See also

- Unmanaged API Reference

Authenticode (Unmanaged API Reference)

09/15/2021

Supports the Authenticode XrML license creation and verification module.

In This Section

_AxlGetIssuerPublicKeyHash Function
Retrieves the SHA-1 hash of the public key associated with the private key that is used to sign the specified certificate.

_AxlPublicKeyBlobToPublicKeyToken Function
Computes the strong name public key token from a CSP PUBLICKEYBLOB format.

_AxlRSAKeyValueToPublicKeyToken Function
Converts a Modulus and Exponent to a strong name public key token.

CertFreeAuthenticodeSignerInfo Function
Frees resources allocated for the AXL_AUTHENTICODE_SIGNER_INFO structure.

CertFreeAuthenticodeTimestamperInfo Function
Frees resources allocated for the AXL_AUTHENTICODE_TIMESTAMPER_INFO structure.

CertTimestampAuthenticodeLicense Function
Time stamps an Authenticode XrML license created by CertCreateAuthenticodeLicense.

CertVerifyAuthenticodeLicense Function
Verifies the validity of an Authenticode XrML license.

AXL_AUTHENTICODE_SIGNER_INFO Structure
Defines the Authenticode signer information.

AXL_AUTHENTICODE_TIMESTAMPER_INFO Structure
Defines the Authenticode time stamper information.

Requirements

Library: clr.dll

See also

- Unmanaged API Reference

_AxlGetIssuerPublicKeyHash Function

Article • 09/15/2021

Retrieves the SHA-1 hash of the public key associated with the private key that is used to sign the specified certificate.

Syntax

```
C++
```

```
HRESULT _AxlGetIssuerPublicKeyHash (
    [in]  IN PCRYPT_DATA_BLOB   pChainContext,
    [out] LPWSTR                *ppwszPublicKeyHash
);
```

Parameters

`pChainContext`

[in] The CSP public key blob. See the CRYPTOAPI_BLOB structure.

`ppwszPublicKeyHash`

[out] A pointer to WCHAR * to receive the hex-encoded public key token.

Return Value

`S_OK` if the function succeeds; otherwise `S_FALSE`.

Requirements

Assembly: clr.dll

See also

- Authenticode

_AxlPublicKeyBlobToPublicKeyToken Function

Article • 09/15/2021

Computes the strong name public key token from a CSP PUBLICKEYBLOB format.

Syntax

```
C++
```

```
HRESULT _AxlPublicKeyBlobToPublicKeyToken (
    [in]  PCCERT_CHAIN_CONTEXT   pCspPublicKeyBlob,
    [out] LPWSTR                 *ppwszPublicKeyToken
);
```

Parameters

`pCspPublicKeyBlob`

[in] The CSP public key blob.

`ppwszPublicKeyHash`

[out] A pointer to WCHAR * to receive the hex-encoded public key hash.

Return Value

`S_OK` if the function succeeds; otherwise `S_FALSE`.

Requirements

Assembly: clr.dll

See also

- Authenticode

_AxlRSAKeyValueToPublicKeyToken function

Article • 09/15/2021

Converts a Modulus and Exponent to a strong name public key token.

Syntax

```
C++
```

```
HRESULT _AxlRSAKeyValueToPublicKeyToken (
    [in]  PCRYPT_DATA_BLOB pModulusBlob,
    [in]  PCRYPT_DATA_BLOB pExponentBlob,
    [out] LPWSTR           *ppwszPublicKeyToken
);
```

Parameters

`pModulusBlob`

[in] The base64-encoded Modulus blob (from the <Modulus> element). See the CRYPTOAPI_BLOB structure.

`pExponentBlob`

[in] The base64-encoded Exponent blob (from the <Exponent> element). See the CRYPTOAPI_BLOB structure.

`ppwszPublicKeyToken`

[out] A pointer to WCHAR * to receive the hex-encoded public key token.

Return Value

`S_OK` if the function succeeds. Otherwise, returns an error code.

Requirements

Assembly: clr.dll

See also

- Authenticode

CertFreeAuthenticodeSignerInfo Function

Article • 09/15/2021

Frees resources allocated for the AXL_AUTHENTICODE_SIGNER_INFO structure.

Syntax

```
C++

HRESULT CertFreeAuthenticodeSignerInfo (
    [in, out]  PAXL_AUTHENTICODE_SIGNER_INFO   pSignerInfo);
```

Parameters

`pSignerInfo`

[in, out] Signer information to be released. See the AXL_AUTHENTICODE_SIGNER_INFO structure.

Return Value

`S_OK` if the function succeeds. Otherwise, returns an error code.

Requirements

Assembly: clr.dll

See also

- Authenticode

CertFreeAuthenticodeTimestamperInfo Function

Article • 09/15/2021

Frees resources allocated for the AXL_AUTHENTICODE_TIMESTAMPER_INFO structure.

Syntax

```
C++

HRESULT CertFreeAuthenticodeTimestamperInfo (
    [in, out]  PAXL_AUTHENTICODE_TIMESTAMPER_INFO   pTimestamperInfo
);
```

Parameters

`pTimestamperInfo`

[in, out] The time stamper information to be released. See the AXL_AUTHENTICODE_TIMESTAMPER_INFO structure.

Return Value

`S_OK` if the function succeeds. Otherwise, returns an error code.

Requirements

Assembly: clr.dll

See also

- Authenticode

CertTimestampAuthenticodeLicense Function

Article • 09/15/2021

Time-stamps an Authenticode XrML license.

Syntax

```
C++
```

```
HRESULT CertTimestampAuthenticodeLicense (
    [in]  PCRYPT_DATA_BLOB   pSignedLicenseBlob,
    [in]  LPCWSTR            pwszTimestampURI,
    [out] PCRYPT_DATA_BLOB   pTimestampSignatureBlob
);
```

Parameters

`pSignedLicenseBlob`

[in] The signed Authenticode XrML license to be time-stamped. See the CRYPTOAPI_BLOB structure.

`pwszTimestampURI`

[in] The time-stamp server's URI.

`pTimestampSignatureBlob`

[out] A pointer to CRYPT_DATA_BLOB to receive the base64-encoded time-stamp signature. It is the caller's responsibility to free `pTimestampSignatureBlob->pbData` with `HepFree()` after use. See the CRYPTOAPI_BLOB structure.

Remarks

The time-stamp signature is actually a PKCS #7 SignedData message whose content is the binary form of the SignatureValue from the license's signature. Basically, this acts as a counter-signature of the license.

Return Value

`S_OK` if the function succeeds. Otherwise, returns an error code.

Requirements

Assembly: clr.dll

See also

- Authenticode

CertVerifyAuthenticodeLicense Function

Article • 09/15/2021

Verifies the validity of an Authenticode XrML license.

Syntax

```
C++
```

```
HRESULT CertVerifyAuthenticodeLicense (
    [in]   PCRYPT_DATA_BLOB                  pLicenseBlob,
    [in]   OPTIONAL DWORD                    dwFlags,
    [out]  PAXL_AUTHENTICODE_SIGNER_INFO     pSignerInfo,
    [out]  PAXL_AUTHENTICODE_TIMESTAMPER_INFO pTimestamperInfo
);
```

Parameters

`pLicenseBlob`

[in] The Authenticode XrML license to be verified.

See the CRYPTOAPI_BLOB structure.

`dwFlags`

[in] Optional. A combination of following values:

- AXL_REVOCATION_NO_CHECK
- AXL_REVOCATION_CHECK_END_CERT_ONLY
- AXL_REVOCATION_CHECK_ENTIRE_CHAIN
- AXL_URL_CACHE_ONLY_RETRIEVAL
- AXL_LIFETIME_SIGNING
- AXL_TRUST_MICROSOFT_ROOT_ONLY

`pSignerInfo`

[out] To receive the signer's information. If the license wasn't signed, `dwError` is set to TRUST_E_NOSIGNATURE. It is the caller's responsibility to free resources by using the CertFreeAuthenticodeSignerInfo function after use.

See AXL_AUTHENTICODE_SIGNER_INFO Structure.

`pTimestamperInfo`
[out] To receive time stamper's information, if available. If the license was not time-stamped, `dwError` is set to TRUST_E_NOSIGNATURE. It is the caller's responsibility to free resources by using the CertFreeAuthenticodeTimestamperInfo function after use.

See AXL_AUTHENTICODE_TIMESTAMPER_INFO Structure.

Return Value

Returns `S_OK` if successful. Otherwise, returns an error code.

Requirements

Assembly: clr.dll

See also

- Authenticode
- GetHashFromHandle Method
- ICLRStrongName Interface

AXL_AUTHENTICODE_SIGNER_INFO Structure

Article • 09/15/2021

Defines the Authenticode signer information.

Syntax

C++

```
typedef struct _AXL_AUTHENTICODE_SIGNER_INFO {
    DWORD cbSize;
    HRESULT dwError;
    ALG_ID algHash;
    LPCWSTR pwszHash
    LPCWSTR pwszDescription;
    LPCWSTR pwszDescriptionUrl;
    PCCERT_CHAIN_CONTEXT pChainContext
} AXL_AUTHENTICODE_SIGNER_INFO, * PAXL_AUTHENTICODE_SIGNER_INFO;
```

Members

Expand table

Member	Description
`cbSize`	The size of this structure.
`dwError`	The error code.
`algHash`	The hash algorithm.
`pwszHash`	The hash.
`pwszDescription`	The description.
`pwszDescriptionUrl`	The URL of the description.
`pChainContext`	The chain context of the signer. See the CERT_CONTEXT structure.

See also

- Authenticode

AXL_AUTHENTICODE_TIMESTAMPER_INFO Structure

Article • 09/15/2021

Defines the Authenticode time stamper information.

Syntax

C++

```
typedef struct _AXL_AUTHENTICODE_SIGNER_INFO {
    DWORD cbSize;
    HRESULT dwError;
    ALG_ID algHash;
    FILETIME ftTimestamp
    PCCERT_CHAIN_CONTEXT pChainContext;
} AXL_AUTHENTICODE_TIMESTAMPER_INFO, * PAXL_AUTHENTICODE_TIMESTAMPER_INFO;
```

Members

Expand table

Member	Description
`cbSize`	The size of this structure.
`dwError`	The error code.
`algHash`	The hash algorithm.
`ftTimestamp`	The time of the time stamp.
`pChainContext`	The time stamper's chain context. See the CERT_CONTEXT structure.

See also

- Authenticode

Constants (Unmanaged API Reference)

Article • 04/20/2024

This topic describes the language type, language vendor, and document type constants that are defined in CorSym.idl.

Language Type Constants

The following table shows language type constants, which represent GUIDs that identify programming languages.

Expand table

Symbol	Description
CorSym_LanguageType_C	Indicates the C language.
CorSym_LanguageType_CPlusPlus	Indicates the C++ language.
CorSym_LanguageType_CSharp	Indicates the C# language.
CorSym_LanguageType_Basic	Indicates the Basic language.
CorSym_LanguageType_Java	Indicates the Java language.
CorSym_LanguageType_Cobol	Indicates the COBOL language.
CorSym_LanguageType_Pascal	Indicates the Pascal language.
CorSym_LanguageType_ILAssembly	Indicates the common intermediate language (CIL) assembly code.
CorSym_LanguageType_JScript	Indicates the JScript language.
CorSym_LanguageType_SMC	Indicates the SMC language.
CorSym_LanguageType_MCPlusPlus	Indicates the C++ language enabled for the .NET Framework.

Language Vendor Constants

The following table shows language vendor constants, which represent GUIDs that identify programming language vendors.

Expand table

Symbol	Description
CorSym_LanguageVendor_Microsoft	Indicates Microsoft.

Document Type Constants

The following table shows document type constants, which represent GUIDs that identify document types.

Expand table

Symbol	Description
CorSym_DocumentType_Text	Indicates a text document.
CorSym_DocumentType_MC	Indicates a non-text document.

See also

- Unmanaged API Reference

Debugging (Unmanaged API Reference)

08/30/2025

The debugging API enables a debugger to debug code that runs in the common language runtime (CLR) environment. The code to be debugged can be any type of code that the CLR supports.

These articles describe APIs for .NET Framework. For unmanaged APIs that are supported on .NET (Core), see .NET debugging (unmanaged API reference).

In This Section

Debugging Coclasses
Describes the unmanaged coclasses that the debugging API uses.

Debugging Interfaces
Describes the unmanaged interfaces that allow debugging of a program that is being executed by the CLR.

Debugging Global Static Functions
Describes the unmanaged global static functions that the debugging API uses.

Debugging Enumerations
Describes the unmanaged enumerations that the debugging API uses.

Debugging Structures
Describes the unmanaged structures that the debugging API uses.

See also

- .NET debugging

Diagnostics Symbol Store (Unmanaged API Reference)

Article • 09/15/2021

The diagnostics symbol store API enables a compiler to generate symbol information for use by a debugger.

In This Section

Diagnostics Symbol Store Interfaces
Describes the unmanaged interfaces that the diagnostics symbol store API uses.

Diagnostics Symbol Store Enumerations
Describes the unmanaged enumerations that the diagnostics symbol store API uses.

Diagnostics Symbol Store Structures
Describes the unmanaged structures that the diagnostics symbol store API uses.

Fusion (Unmanaged API Reference)

Article • 09/15/2021

The fusion API enables a runtime host to access the properties of an application's resources in order to locate the correct versions of those resources for the application.

In This Section

Fusion Interfaces

Describes the unmanaged interfaces that the fusion API uses.

Fusion Global Static Functions

Describes the unmanaged global static functions that the fusion API uses.

Fusion Enumerations

Describes the unmanaged enumerations that the fusion API uses.

Fusion Structures

Describes the unmanaged structures that the fusion API uses.

Hosting (Unmanaged API Reference)

Article • 09/15/2021

The hosting API enables unmanaged hosts to integrate the common language runtime (CLR) into their applications.

In This Section

Hosting Coclasses
Describes the unmanaged coclasses that the hosting API uses.

Hosting Enumerations
Describes the unmanaged enumerations that the hosting API uses.

Hosting Global Static Functions
Describes the unmanaged global static functions that the hosting API uses.

Hosting Interfaces
Describes the unmanaged interfaces that enable a runtime host to integrate the CLR into an unmanaged application.

Hosting Structures
Describes the unmanaged structures that the hosting API uses.

Related Sections

Runtime Hosts
Describes the runtime hosts included with the .NET Framework.

Metadata (Unmanaged API Reference)

08/30/2025

The metadata API enables a client, such as a compiler, to generate or access a component's metadata without the types being loaded by the common language runtime (CLR).

These articles describe APIs for .NET Framework. For unmanaged APIs that are supported on .NET (Core), see .NET metadata (unmanaged API reference).

In This Section

Metadata Interfaces
Describes the unmanaged interfaces that provide access to the metadata exposed by the .NET Framework types, methods, fields, and so on.

Metadata Global Static Functions
Describes the unmanaged global static functions that the metadata API uses.

Metadata Enumerations
Describes the unmanaged enumerations that the metadata API uses.

Metadata Structures
Describes the unmanaged structures that the metadata API uses.

Metadata Unions
Describes the unmanaged unions that the metadata API uses.

Profiling (Unmanaged API Reference)

08/30/2025

The profiling API enables a profiler to monitor a program's execution by the common language runtime (CLR).

These articles describe APIs for .NET Framework. For unmanaged APIs that are supported on .NET (Core), see .NET profiling (unmanaged API reference).

In This Section

Profiling Overview Describes the services and interfaces that the CLR provides to support profiling in the .NET Framework environment.

Profiling Interfaces Describes the unmanaged interfaces that the profiling API uses.

Setting Up a Profiling Environment Describes the steps you must take to profile a .NET Framework application.

CLR Profilers and Windows Store Apps Discusses how to port diagnostic tools that consume the CLR Profiling API to work successfully with Windows Store apps.

CORPROF_E_UNSUPPORTED_CALL_SEQUENCE HRESULT Documents the conditions under which a method call returns the `CORPROF_E_UNSUPPORTED_CALL_SEQUENCE` HRESULT.

Profiling Global Static Functions Describes the unmanaged global static functions that the profiling API uses.

Profiling Enumerations Describes the unmanaged enumerations that the profiling API uses.

Profiling Structures Describes the unmanaged structures that the profiling API uses.

Strong Naming (Unmanaged API Reference)

Article • 09/15/2021

The strong naming API enables a client to administer strong name signing for assemblies.

Signing an assembly with a strong name adds a public key encryption to the file containing the assembly manifest. Strong name signing helps verify name uniqueness, prevents name spoofing, and provides callers with a unique identity when a reference is resolved. However, no level of trust is associated with a strong name.

In This Section

> **Note**
>
> All of these functions have been deprecated starting with the .NET Framework 4. For suggested alternatives, see the ICLRStrongName interface.

GetHashFromAssemblyFile Function
Gets a hash of the specified assembly file, using the specified hash algorithm. Deprecated starting with the .NET Framework 4.

GetHashFromAssemblyFileW Function
Gets a hash of the assembly file specified as a Unicode string, using the specified hash algorithm. Deprecated starting with the .NET Framework 4.

GetHashFromBlob Function
Gets a hash of the assembly at the specified memory address, using the specified hash algorithm. Deprecated starting with the .NET Framework 4.

GetHashFromFile Function
Generates a hash over the contents of the specified file. Deprecated starting with the .NET Framework 4.

GetHashFromFileW Function
Generates a hash over the contents of the file specified by a Unicode string. Deprecated starting with the .NET Framework 4.

GetHashFromHandle Function
Generates a hash over the contents of the file with the specified file handle, using the specified hash algorithm. Deprecated starting with the .NET Framework 4.

StrongNameCompareAssemblies Function
Determines whether two assemblies differ only by their strong name signatures. Deprecated starting with the .NET Framework 4.

StrongNameErrorInfo Function
Gets the last error code that was raised by one of the strong name functions.

StrongNameFreeBuffer Function
Frees memory that was allocated with a previous call to a strong name function such as StrongNameGetPublicKey, StrongNameTokenFromPublicKey, or StrongNameSignatureGeneration. Deprecated starting with the .NET Framework 4.

StrongNameGetBlob Function
Fills the specified buffer with the binary representation of the executable file at the specified address. Deprecated starting with the .NET Framework 4.

StrongNameGetBlobFromImage Function
Gets a binary representation of the assembly image at the specified memory address. Deprecated starting with the .NET Framework 4.

StrongNameGetPublicKey Function
Gets the public key from a private/public key pair. Deprecated starting with the .NET Framework 4.

StrongNameHashSize Function
Gets the buffer size required for a hash, using the specified hash algorithm. Deprecated starting with the .NET Framework 4.

StrongNameKeyDelete Function
Deletes the specified key container. Deprecated starting with the .NET Framework 4.

StrongNameKeyGen Function
Creates a new public/private key pair for strong name use. Deprecated starting with the .NET Framework 4.

StrongNameKeyGenEx Function
Generates a new public/private key pair with the specified key size for strong name use. Deprecated starting with the .NET Framework 4.

StrongNameKeyInstall Function
Imports a public/private key pair into a container. Deprecated starting with the .NET

Framework 4.

StrongNameSignatureGeneration Function
Generates a strong name signature for the specified assembly. Deprecated starting with the .NET Framework 4.

StrongNameSignatureGenerationEx Function
Generates a strong name signature for the specified assembly, based on the specified flags. Deprecated starting with the .NET Framework 4.

StrongNameSignatureSize Function
Returns the size of the strong name signature. Deprecated starting with the .NET Framework 4.

StrongNameSignatureVerification Function
Gets a value indicating whether the assembly manifest at the supplied path contains a strong name signature, which is verified according to the specified flags. Deprecated starting with the .NET Framework 4.

StrongNameSignatureVerificationEx Function
Gets a value indicating whether the assembly manifest at the supplied path contains a strong name signature. Deprecated starting with the .NET Framework 4.

StrongNameSignatureVerificationFromImage Function
Verifies that an assembly that has already been mapped to memory is valid for the associated public key. Deprecated starting with the .NET Framework 4.

StrongNameTokenFromAssembly Function
Creates a strong name token from the specified assembly file. Deprecated starting with the .NET Framework 4.

StrongNameTokenFromAssemblyEx Function
Creates a strong name token from the specified assembly file, and returns the public key. Deprecated starting with the .NET Framework 4.

StrongNameTokenFromPublicKey Function
Gets a token representing a public key. Deprecated starting with the .NET Framework 4.

PublicKeyBlob Structure
Represents the public key of a public/private key pair in binary format.

See also

- ICLRStrongName Interface

- Unmanaged API Reference

Windows Management Instrumentation (WMI) and Performance Counters (Unmanaged API Reference)

Article • 09/15/2021

The .NET Framework WMI and Performance Counters unmanaged API consists of a set of functions that wrap calls to the native Windows Management Instrumentation API. It allows you to develop tools and libraries that manage and monitor remote computer systems.

> **Note**
>
> This API is for internal use only. It's not intended for use from developer code.

The API includes the following functions:

Expand table

Function	Description
BeginEnumeration function	Resets the enumerator to the beginning of an enumeration of WMI object properties.
BeginMethodEnumeration function	Begins an enumeration of the methods available for an object.
BlessIWbemServices function	Indicates whether the user credentials permit access to a specified IWbemServices class.
BlessIWbemServicesObject function	Indicates whether the user credentials permit access to a specified IWbem service object.
Clone function	Returns a new object that is a complete clone of the current object.
CloneEnumWbemClassObject function	Makes a logical copy of an enumerator, retaining its current position in an enumeration.
CompareTo function	Compares an object to another Windows management object.
ConnectServerWmi function	Creates a connection through DCOM to a WMI namespace on a specified computer.

Function	Description
CreateClassEnumWmi function	Returns an enumerator for all classes that satisfy the specified selection criteria.
CreateInstanceEnumWmi function	Returns an enumerator that returns the instances of a specified class that meet specified selection criteria.
Delete function	Deletes a specified property from a class definition and all of its qualifiers.
DeleteMethod function	Deletes a specified method from a CIM class definition.
EndEnumeration function	Terminates an enumeration sequence.
EndMethodEnumeration function	Terminates an enumeration sequence started by calling the BeginMethodEnumeration function.
ExecNotificationQueryWmi function	Executes a query to receive events.
ExecQueryWmi function	Executes a query to retrieve objects.
FormatFromRawValue function	Converts one raw performance data value to the specified format, or two raw performance data values if the format conversion is time-based.
Get function	Retrieves a specified property value if it exists.
GetCurrentApartmentType function	Retrieves the type of apartment in which the caller is executing.
GetDemultiplexedStub function	Creates an object forwarder sink to assist a client in receiving asynchronous calls from Windows Management.
GetErrorInfo function	Retrieves error information from the previous function call.
GetMethod function	Retrieves information about the specified method.
GetMethodOrigin function	Determines the class in which a method is declared.
GetMethodQualifierSet function	Retrieves the qualifier set for a particular method.
GetNames function	Retrieves either a subset or all of the names of the properties of an object.
GetObjectText function	Returns a textual rendering of an object in the MOF syntax.
GetPropertyHandle function	Returns a unique handle that identifies a property.
GetPropertyOrigin function	Determines the class in which a property is declared.
GetPropertyQualifierSet function	Retrieves the qualifier set for a particular property.

Function	Description
GetQualifierSet function	Retrieves the qualifier set for a class instance or a class definition.
InheritsFrom function	Determines whether the current class or instance derives from a specified parent class.
Initialize function	Performs WMI initialization.
Next function	Retrieves the next property in an enumeration.
NextMethod function	Retrieves the next method in an enumeration.
Put function	Sets a named property to a new value.
PutClassWmi function	Creates a new class or updates an existing one.
PutInstanceWmi function	Creates or updates an instance of an existing class. The instance is written to the WMI repository.
PutMethod function	Creates a method.
QualifierSet_BeginEnumeration function	Resets an enumerator of the qualifiers of an object to the beginning of the enumeration.
QualifierSet_Delete function	Deletes a specified qualifier by name.
QualifierSet_EndEnumeration function	Terminates the enumeration begun with a call to the `QualifierSet_BeginEnumeration` function.
QualifierSet_Get function	Gets the specified named qualifier.
QualifierSet_GetNames function	Retrieves the names of all qualifiers or of specified qualifiers that are available from the current object or property.
QualifierSet_Next function	Retrieves the next qualifier in an enumeration that started with a call to the QualifierSet_BeginEnumeration function.
QualifierSet_Put function	Writes the named qualifier and value.
ResetSecurity function	Assigns the supplied impersonation token to the current thread.
SetSecurity function	Retrieves the impersonation token associated with the current thread.
SpawnDerivedClass function	Creates a newly derived class object from a specified object.
SpawnInstance function	Creates a new instance of a class.
VerifyClient function	Ensures that the client key has the correct security.

Function	Description
WritePropertyValue function	Writes a specified number of bytes to a property identified by a property handle.

See also

- Unmanaged API reference

Tlbexp Helper Functions (Unmanaged API Reference)

Article • 09/15/2021

The Type Library Exporter tool (Tlbexp.exe) loads a dynamic link library named TlbRef.dll. This DLL contains two helper functions and an interface that the exporter tool uses during the assembly-to-type-library conversion process.

In This Section

GetTypeLibInfo Function
Provides localization and operating system information for a type library.

LoadTypeLibWithResolver Function
Loads a type library by using an implementation of the ITypeLibResolver interface to resolve any referenced type libraries.

ITypeLibResolver Interface
Provides the ResolveTypeLib method, which returns the fully qualified path of a type library.

GetTypeLibInfo Function

Article • 09/15/2021

Returns information about the specified type library by examining its TLIBATTR structure.

Syntax

C++

```
HRESULT GetTypeLibInfo(
    [in]   LPWSTR     szFile,
    [out]  GUID      *pTypeLibID,
    [out]  LCID      *pTypeLibLCID,
    [out]  SYSKIND   *pTypeLibPlatform,
    [out]  USHORT    *pTypeLibMajorVer,
    [out]  USHORT    *pTypeLibMinorVer
);
```

Parameters

`szFile`

[in] The file name of the type library.

`pTypeLibID`

[out] The GUID of the type library.

`pTypeLibLCID`

[out] The localization ID of the type library.

`pTypeLibPlatform`

[out] A SYSKIND flag that identifies the target operating system for the type library. Common values are SYS_WIN32 and SYS_WIN64.

`pTypeLibMajorVer`

[out] The major version number of the type library. For example, for version *x.y*, the major version number is *x*.

`pTypeLibMinorVer`

[out] The minor version number of the type library. For example, for version *x.y*, the minor version number is *y*.

Remarks

The `GetTypeLibInfo` function is called by the Tlbexp.exe (Type Library Exporter). This tool generates a type library that describes the types in a common language runtime (CLR) assembly.

If any parameter is null, the function returns an `HRESULT` of `E_POINTER`. Otherwise, it returns `S_OK`.

Requirements

Platforms: See System Requirements.

Header: TlbRef.h

Library: TlbRef.lib

.NET Framework Versions: Available since 2.0

See also

- Tlbexp Helper Functions
- LoadTypeLibEx Function

LoadTypeLibWithResolver Function

Article • 11/30/2021

Loads a type library and uses the supplied ITypeLibResolver interface to resolve any internally referenced type libraries.

Syntax

```
C++

HRESULT LoadTypeLibWithResolver(
    [in]  LPCOLESTR           szFile,
    [in]  REGKIND             regkind,
    [in]  ITypeLibResolver   *pTlbResolver,
    [out] ITypeLib          **pptlib);
```

Parameters

`szFile`

[in] The file path of the type library.

`regkind`

[in] A REGKIND enumeration flag that controls how the type library is registered. Its possible values are:

- `REGKIND_DEFAULT`: Use default registration behavior.
- `REGKIND_REGISTER`: Register this type library.
- `REGKIND_NONE`: Do not register this type library.

`pTlbResolver`

[in] A pointer to the implementation of the ITypeLibResolver interface.

`pptlib`

[out] A reference to the type library that is being loaded.

Return Value

One of the HRESULT values listed in the following table.

Expand table

Return value	Meaning
`S_OK`	Success.
`E_OUTOFMEMORY`	Out of memory.
`E_POINTER`	One or more of the pointers are invalid.
`E_INVALIDARG`	One or more of the arguments are invalid.
`TYPE_E_IOERROR`	The function could not write to the file.
`TYPE_E_REGISTRYACCESS`	The system registration database could not be opened.
`TYPE_E_INVALIDSTATE`	The type library could not be opened.
`TYPE_E_CANTLOADLIBRARY`	The type library or DLL could not be loaded.

Remarks

The Tlbexp.exe (Type Library Exporter) calls the `LoadTypeLibWithResolver` function during the assembly-to-type-library conversion process.

This function loads the specified type library with minimal access to the registry. The function then examines the type library for internally referenced type libraries, each of which must be loaded and added to the parent type library.

Before a referenced type library can be loaded, its reference file path must be resolved to a full file path. This is accomplished through the ResolveTypeLib method that is provided by the ITypeLibResolver interface, which is passed in the `pTlbResolver` parameter.

When the full file path of the referenced type library is known, the `LoadTypeLibWithResolver` function loads and adds the referenced type library to the parent type library, creating a combined primary type library.

After the function resolves and loads all internally referenced type libraries, it returns a reference to the primary resolved type library in the `pptlib` parameter.

The `LoadTypeLibWithResolver` function is generally called by the Tlbexp.exe (Type Library Exporter), which supplies its own internal ITypeLibResolver interface implementation in the `pTlbResolver` parameter.

If you call `LoadTypeLibWithResolver` directly, you must supply your own ITypeLibResolver interface implementation.

Requirements

Platforms: See System Requirements.

Header: TlbRef.h

Library: TlbRef.lib

.NET Framework Version: 3.5, 3.0, 2.0

See also

- Tlbexp Helper Functions
- LoadTypeLibEx Function

ITypeLibResolver Interface

Article • 09/15/2021

Provides the ResolveTypeLib method, which resolves the file path of a type library.

Methods

Expand table

Method	Description
ResolveTypeLib Method	Resolves the simple name of a type library by returning its fully qualified path.

Requirements

Platforms: See System Requirements.

Header: TlbRef.idl, TlbRef.h

Library: TlbRef.lib

.NET Framework Version: 4, 3.5, 3.0, 2.0

See also

- Tlbexp Helper Functions
- LoadTypeLibEx function

ResolveTypeLib Method

Article • 09/15/2021

Resolves the simple name of a type library by returning its fully qualified path.

Syntax

```
C++

HRESULT ResolveTypeLib(
    [in]  BSTR      bstrSimpleName,
    [in]  GUID      tlbid,
    [in]  LCID      lcid,
    [in]  USHORT    wMajorVersion,
    [in]  USHORT    wMinorVersion,
    [in]  SYSKIND   syskind,
    [out] BSTR     *pbstrResolvedTlbName);
```

Parameters

`bstrSimpleName`

[in] A BSTR that contains the simple name of the type library.

`tlbid`

[in] The GUID assigned to the type library in the registry.

`lcid`

[in] The localization ID of the type library.

`wMajorVersion`

[in] The major version number of the type library. For example, for version *x.y*, the major version number is *x*.

`wMinorVersion`

[in] The minor version number of the type library. For example, for version *x.y*, the minor version number is *y*.

`syskind`

[in] A SYSKIND flag that identifies the operating environment. Common values are SYS_WIN32 and SYS_WIN64.

`pbstrResolvedTlbName`

[out] A pointer to a BSTR that contains the full path of the type library named in the `bstrSimpleName` parameter.

Remarks

The `ResolveTypeLib` method is called by the LoadTypeLibWithResolver function during Tlbexp.exe (Type Library Exporter) processing.

Custom implementations of this interface must return a BSTR that contains the full path of the type library named in the `bstrSimpleName` parameter.

Requirements

Platforms: See System Requirements.

Header: TlbRef.idl, TlbRef.h

Library: TlbRef.lib

.NET Framework Versions: Available since 2.0

See also

- Tlbexp Helper Functions
- LoadTypeLibEx

GUID_ManagedName Attribute

Article • 09/15/2021

Defines a custom interface attribute that specifies the managed namespace name for a component object model (COM) library.

Syntax

idl

```
[
   custom(GUID_ManagedName, value)
]
```

Parameters

`value`

The managed namespace name for the library.

Definition

`GUID_ManagedName` is defined in Cor.h as follows:

C++

```
// {0F21F359-AB84-41e8-9A78-36D110E6D2F9}
EXTERN_GUID(GUID_ManagedName, 0xf21f359, 0xab84, 0x41e8, 0x9a, 0x78, 0x36,
0xd1, 0x10, 0xe6, 0xd2, 0xf9);
```

Remarks

A custom interface attribute defines metadata for an object in the type library.

Use ITypeInfo2.GetCustData or ITypeLib2.GetCustData to retrieve the managed name from the attribute.

For more information, see Interface Attributes in the Visual C++ reference documentation.

Example

The following example shows a library definition using the `GUID_ManagedName` attribute.

idl

```
[
   ...
   custom(GUID_ManagedName, Microsoft.VisualStudio.CommandBars.dll")
]
library Microsoft_VisualStudio_CommandBars
{
   ...
}
```

Requirements

Header: Cor.h