NGWS

Unmanaged Symbolic Info Interfaces

This is preliminary documentation and subject to change

Last update: 8 June 2000

Table of Contents

1 Overview

Symbolic information is often used by tools such as debuggers and profilers. This document describes a set of unmanaged interfaces that can be used by tool developers to read or write symbolic information.

2 Unmanaged Classes

2.1.1 CorSymBinder

The **CorSymBinder** class is the default implementation of the **ISymUnmanagedBinder** interface.

IDL Declaration

```
coclass CorSymBinder
{
   [default] interface ISymUnmanagedBinder;
};
```

2.1.2 CorSymReader

The **CorSymReader** class is the default implementation of the **ISymUnmanagedBinder** interface that targets the PDB symbolic information format.

IDL Declaration

```
coclass CorSymReader
{
   [default] interface ISymUnmanagedReader;
};
```

2.1.3 CorSymWriter

The **CorSymWriter** class is the default implementation of the **ISymUnmanagedWriter** interface that targets the PDB symbolic information format.

IDL Declaration

```
coclass CorSymWriter
{
   [default] interface ISymUnmanagedWriter;
};
```

3 API

3.1.1 ISymUnmanagedBinder

The **ISymUnmanagedBinder** interface is used to get the interface for the symbol reader for this module. The class **CoSymBinder** is the Microsoft implementation of this class.

IDL Declaration

```
interface ISymUnmanagedBinder : IUnknown
{
   HRESULT GetReaderForFile([in] IUnknown* importer, [in] const WCHAR*
fileName, [in] const WCHAR *searchPath, [out, retval]
ISymUnmanagedReader** pRetVal);
   HRESULT GetReaderFromStream([in] IUnknown* importer, [in] IStream
*pIStream, [out, retval] ISymUnmanagedReader** pRetVal);
};
```

GetReaderForFile

```
HRESULT GetReaderForFile([in] IUnknown* importer, [in] const WCHAR*
fileName, [in] const WCHAR *searchPath, [out, retval]
ISymUnmanagedReader** pRetVal);
```

Given a metadata interface and a file name, returns the correct **ISymUnmanagedReader** interface that will read the debugging symbols associated with the module.

Parameter	Description
importer	Pointer to the metadata import interface.
fileName	Pointer to the name of the file for which the unmanaged reader interface is required.
searchPath	Pointer to the search path used to locate the symbol file.
pRetVal	The **ISymUnmanagedReader** interface that will read the debugging symbols.

GetReaderFromStream

```
HRESULT GetReaderModule([in] IUnknown* importer, [in] IStream
*pIStream, [out, retval] ISymUnmanagedReader** pRetVal);
```

Given a metadata interface, returns the correct **ISymUnmanagedReader** interface that will read the debugging symbols associated with the given IStream.

Parameter	Description
importer	Pointer to the metadata import interface.

pIStream	Pointer to the stream containing the symbol store.
pRetVal	The **ISymUnmanagedReader** interface that will read the debugging symbols.

3.1.2 ISymUnmanagedDocument

The **ISymUnmanagedDocument** interface represents a document referenced by a symbol store. A document is defined by a URL and a document type GUID. Using the document type GUID and the URL, one can locate the document however it is stored. Document source can optionally be stored in the symbol store. This interface also provides access to that source if it is present.

IDL Declaration

```
interface ISymUnmanagedDocument : IUnknown
{
   HRESULT FindClosestLine([in] ULONG32 line, [out, retval] ULONG32*
pRetVal);
   HRESULT GetCheckSum([in] ULONG32 cData, [out] ULONG32 *pcData, [out]
BYTE data[]);
   HRESULT GetCheckSumAlgorithmId([out, retval] GUID* pRetVal);
   HRESULT GetDocumentType([out, retval] GUID* pRetVal);
   HRESULT GetLanguageVendor([out, retval] GUID* pRetVal);
   HRESULT GetLanguage([out, retval] GUID* pRetVal);
   HRESULT GetSourceLength([out, retval] ULONG32* pRetVal);
   HRESULT GetSourceRange([in] ULONG32 startLine, [in] ULONG32
startColumn, [in] ULONG32 endLine, [in] ULONG32 endColumn, [in] ULONG32
cSourceBytes, [out] *pcSourceBytes, [out] BYTE source[]);
   HRESULT GetURL([in] ULONG32 cchUrl, [out] ULONG32 *pcchUrl, [out]
WCHAR szUrl[]);
   HRESULT HasEmbeddedSource([out, retval] BOOL* pRetVal);
};
```

FindClosestLine

```
HRESULT FindClosestLine([in] ULONG32 line, [out, retval] ULONG32*
pRetVal);
```

Given a line in this document that may or may not be a sequence point, gets the closest line that is a sequence point.

Parameter	Description
line	The specified line in the document.
pRetVal	The closest line that is a sequence point.

GetCheckSum

```
HRESULT GetCheckSum([in] ULONG32 cData, [out] ULONG32 *pcData, [out]
BYTE data[]);
```

Gets the check sum.

Parameter	Description
cData	The allocated size of the buffer
pcData	The number of bytes available for return. But no more than cData bytes are acutally returned into the buffer
data	The buffer to receive the checksum

GetCheckSumAlgorithimId

```
HRESULT GetCheckSumAlgorithmId([out, retval] GUID* pRetVal);
```

Gets the check sum algorithm id. Return a GUID of all zeros if there is no checksum.

Parameter	Description
pRetVal	The GUID that represents the check sum algorithm id.

GetDocumentType

```
HRESULT GetDocumentType([out, retval] GUID* pRetVal);
```

Gets the type of this document.

Parameter	Description
pRetVal	The type of this document.

GetLanguageVendor

```
HRESULT GetLanguageVendor([out, retval] GUID* pRetVal);
```

Gets the language vendor of this document.

Parameter	Description
pRetVal	The language vendor of this document.

GetLanguage

```
HRESULT GetLanguage([out, retval] GUID* pRetVal);
```

Gets the language of this document.

Parameter	Description
pRetVal	The language of this document.

GetSourceLength

```
HRESULT GetSourceLength([out, retval] ULONG32* pRetVal);
```

Gets the length, in bytes, of the embedded source.

Parameter	Description
pRetVal	The source length of this document.

GetSourceRange

```
HRESULT GetSourceRange([in] ULONG32 startLine, [in] ULONG32
startColumn, [in] ULONG32 endLine, [in] ULONG32 endColumn, [in] ULONG32
cSourceBytes, [out] *pcSourceBytes, [out] BYTE source[]);
```

Gets the embedded document source for the specified range. The buffer must be large enough to hold the source.

Parameter	Description
startLine	The starting line in this document.
startColumn	The starting column in this document.
endLine	The ending line in this document.
endColumn	The ending column in this document.
cSourceBytes	The allocated size of the buffer.
pcSourceBytes	The length of the string returned in the buffer.
source	The string buffer.

GetURL

```
HRESULT GetURL([in] ULONG32 cchUrl, [out] ULONG32 *pcchUrl, [out] WCHAR
szUrl[]);
```

Gets the URL of this document.

Parameter	Description
cchUrl	The allocated size of the buffer
pcchUrl	The number of characters available for return. But no more than cchUrl are actually returned in the buffer
szUrl	The string buffer

HasEmbeddedSource

```
HRESULT HasEmbeddedSource([out, retval] BOOL* pRetVal);
```

Checks if this document is stored in the symbol store.

Parameter	Description
pRetVal	Value is **true** if this document is stored in the symbol store,

	otherwise **false**.

3.1.3 ISymUnmanagedDocumentWriter

The **ISymUnmanagedDocumentWriter** interface provides an interface for writing to a document referenced by a symbol store. A document is defined by a URL and a document type GUID. Document source can optionally be stored in the symbol store.

IDL Declaration

```
interface ISymUnmanagedDocumentWriter : IUnknown
{
   HRESULT SetCheckSum([in] GUID algorithmId, [in] ULONG32
checkSumSize, [in] BYTE checkSum[]);
   HRESULT SetSource([in] ULONG32 sourceSize, [in] BYTE source[]);
};
```

SetCheckSum

```
HRESULT SetCheckSum([in] GUID algorithmId, [in] ULONG32 checkSumSize,
[in] BYTE checkSum[]);
```

Sets check sum information.

Parameter	Description
algorithmId	The GUID representing the algorithm ID.
checkSumSize	The size of the checkSum array.
checkSum	The check sum.

SetSource

```
HRESULT SetSource([in] ULONG32 sourceSize, [in] BYTE source[]);
```

Sets the embedded source for a document being written.

Parameter	Description
sourceSize	The length of the document in bytes.
source	The document source.

3.1.4 ISymUnmanagedMethod

The **ISymUnmanagedMethod** interface represents a method within a symbol reader. This provides access to only the symbol related attributes of a method such as sequence points, lexical scopes, and parameter information. Use it in conjunction with other means such as the System.Reflection classes to read the type related attributes of a method

IDL Declaration

```
interface ISymUnmanagedMethod : IUnknown
{
   HRESULT GetNamespace([out, retval] ISymUnmanagedNamespace
**ppRetVal);
   HRESULT GetOffset([in] ISymUnmanagedDocument* document, [in] ULONG32
line, [in] ULONG32 column, [out, retval] ULONG32* pRetVal);
   HRESULT GetParameters([in] ULONG32 cParams, [out] ULONG32 *pcParams,
[out] ISymUnmanagedVariable* params[]);
   HRESULT GetRanges([in] ISymUnmanagedDocument* document, [in] ULONG32
line, [in] ULONG32 column, [in] ULONG32 cRanges, [out] ULONG32
*pcRanges, [out] ULONG32 ranges[]);
   HRESULT GetRootScope([out, retval] ISymUnmanagedScope** pRetVal);
   HRESULT GetScopeFromOffset([in] ULONG32 offset, [out, retval]
ISymUnmanagedScope** pRetVal);
   HRESULT GetSequencePoints([in] ULONG32 cPoints, [out] ULONG32
*pcPoints, [in] ULONG32 offsets[], [in] ISymUnmanagedDocument*
documents[], [in] ULONG32 startLines[], [in] ULONG32 startColumns[],
[in] ULONG32 endLines[], [in] ULONG32 endColumns[]);
   HRESULT GetSequencePointCount([out, retval] ULONG32* pRetVal);
   HRESULT GetSourceStartEnd([in] ISymUnmanagedDocument *docs[2], [in]
ULONG32 lines[2], [in] ULONG32 columns[2], [out] BOOL *pRetVal);
   HRESULT GetToken([out, retval] mdMethodDef *pRetVal);
};
```

GetNamespace

```
HRESULT GetNamespace([out, retval] ISymUnmanagedNamespace **ppRetVal);
```

Gets the namespace that this method is defined in.

Parameter	Description
ppRetVal	The namespace that this method is defined in.

GetOffset

```
HRESULT GetOffset([in] ISymUnmanagedDocument* document, [in] ULONG32
line, [in] ULONG32 column, [out, retval] ULONG32* pRetVal);
```

Given a position in a document, gets the offset within the method that corresponds to the position.

Parameter	Description
document	The document for which the offset is requested.
line	The document line corresponding to the offset.

column	The document column corresponding to the offset.
pRetVal	The offset within the specified document.

GetParameters

```
HRESULT GetParameters([in] ULONG32 cParams, [out] ULONG32 *pcParams,
[out] ISymUnmanagedVariable* params[]);
```

Get the parameters for this method. The parameters are returned in the order they are defined within the method's signature.

Parameter	Description
cParams	The size of the allocated array.
pcParams	The number of parameters available for return. But no more than cParams are acutally returned in the array
params	The parameter array

GetRanges

```
HRESULT GetRanges([in] ISymUnmanagedDocument* document, [in] ULONG32
line, [in] ULONG32 column, [in] ULONG32 cRanges, [out] ULONG32
*pcRanges, [out] ULONG32 ranges[]);
```

Given a position in a document, get an array of start/end offset pairs that correspond to the ranges of IL that the position covers within this method. The array is an array of integers and is [start, end, start, end]. The number of range pairs is the length of the array divided by 2.

Parameter	Description
document	The document for which the offset is requested.
line	The document line corresponding to the ranges.
column	The document column corresponding to the ranges.
cRanges	The size of the allocated ranges[] array.
pcRanges	The number of ranges available for return. But no more than cRanges are actually returned in the array
ranges	The range array

GetRootScope

```
HRESULT GetRootScope([out, retval] ISymUnmanagedScope** pRetVal);
```

Gets the root lexical scope for this method. This scope encloses the entire method.

Parameter	Description
pRetVal	The root lexical scope for this method.

GetScopeFromOffset

```
HRESULT GetScopeFromOffset([in] ULONG32 offset, [out, retval]
ISymUnmanagedScope** pRetVal);
```

Given an offset within the method, returns the most enclosing lexical scope. This can be used to start local variable searches.

Parameter	Description
offset	The offset, in bytes, from the beginning of the method
pRetVal	The most enclosing lexical scope for the given byte offset within the method

GetSequencePoints

```
HRESULT GetSequencePoints([in] ULONG32 cPoints, [out] ULONG32
*pcPoints, [in] ULONG32 offsets[], [in] ISymUnmanagedDocument*
documents[], [in] ULONG32 startLines[], [in] ULONG32 startColumns[],
[in] ULONG32 endLines[], [in] ULONG32 endColumns[]);
```

Get the sequence points for this method.

Parameter	Description
cPoints	The size of the allocated arrays
pcPoints	The number of sequence points available for return. But no more than cPoints are actually returned in the arrays
offsets	The array of offsets of the sequence points
documents	The array of documents in which the sequence points are located
startLines	The array of lines in the documents at which the sequence points are located
startColumns	The array of columns in the documents at which the sequence points are located
endLines	The array of lines in the documents at which the sequence points end
endColumns	The array of columns in the documents at which the sequence points end

GetSequencePointCount

```
HRESULT GetSequencePointCount([out, retval] ULONG32* pRetVal);
```

Gets the count of the sequence points in the method.

Parameter	Description
pRetVal	The count of the sequence points in the method.

GetSourceStartEnd

```
HRESULT GetSourceStartEnd([in] ISymUnmanagedDocument *docs[2], [in]
ULONG32 lines[2], [in] ULONG32 columns[2], [out] BOOL *pRetVal);
```

Get the start/end positions for the source of this method. The first array position is the start while the second is the end.

Parameter	Description
docs	The starting and ending source documents.
lines	The starting and ending lines in the corresponding source documents.
columns	The starting and ending columns in the corresponding source documents.
pRetVal	True if the positions were defined, false otherwise.

GetToken

```
HRESULT GetToken([out, retval] mdMethodDef pRetVal);
```

Gets the Metadata token for this method.

Parameter	Description
pRetVal	The Metadata token for this method.

3.1.5 ISymUnmanagedNamespace

The **ISymUnmanagedNamespace** interface represents a namespace.

IDL Declaration

```
interface ISymUnmanagedScope : IUnknown
{
   HRESULT GetName([in] ULONG32 cchName, [out] ULONG32 *pcchName, [out]
WCHAR szName[]);
   HRESULT GetNamespaces([in] ULONG32 cNamespaces, [out] ULONG32
*pcNamespaces, [out] ISymUnmanagedNamespaces* namespaces[]);
   HRESULT GetVariables([in] ULONG32 cVars, [out] ULONG32 *pcVars,
[out] ISymUnmanagedVariable* pVars[]);
};
```

GetName

```
HRESULT GetName([in] ULONG32 cchName, [out] ULONG32 *pcchName, [out]
WCHAR szName[]);
```

Gets the name of the namespace.

Parameter	Description
cchName	The size of the allocated array.
pcchName	The number of characters available for return. But no more than cchName are actually returned in the buffer
szName	The name

GetNamespaces

```
HRESULT GetNamespaces([in] ULONG32 cNamespaces, [out] ULONG32
*pcNamespaces, [out] ISymUnmanagedNamespaces* pNamespaces[]);
```

Gets the children of this namespace.

Parameter	Description
cNamespaces	The size of the allocated array.
pcNamespaces	The number of namespaces available for return. But no more than cNamespaces are actually returned in the array
pNamespaces	The namespaces

GetVariables

```
HRESULT GetVariables([in] ULONG32 cVars, [out] ULONG32 *pcVars, [out]
ISymUnmanagedVariable* pVars[]);
```

Gets all the variables defined at global scope within this namespace.

Parameter	Description
cVars	The size of the allocated array.
pcVars	The number of variables available for return. But no more than cVars are actually returned in the array
pVars	The variables

3.1.6 ISymUnmanagedReader

The **ISymUnmanagedReader** interface represents a symbol reader for managed code. The interface provides access to documents, methods, and variables. The class CoSymReader is the Microsoft implementation of this class that targets the PDB symbolic information format.

IDL Declaration

```
interface ISymReader : IUnknown
{
   HRESULT GetDocuments([in] ULONG32 cDocs, [out] ULONG32 *pcDocs,
[out] ISymUnmanagedDocument *pDocs[]);
```

```
   HRESULT GetDocument([in] WCHAR *url, [in] Guid language, [in] Guid
languageVendor, [in] Guid documentType, [out, retval]
ISymUnmanagedDocument** pRetVal);
   HRESULT GetGlobalVariables([in] ULONG32 cVars, [out] ULONG32
*pcVars, [out] ISymUnmanagedVariable *pVars[]);
   HRESULT GetMethod([in] mdMethodDef token, [out, retval]
ISymUnmanagedMethod** pRetVal);
   HRESULT GetMethodByVersion([in] mdMethodDef token, [in] int version,
[out, retval] ISymUnmanagedMethod** pRetVal);
   HRESULT GetMethodFromDocumentPosition([in] ISymUnmanagedDocument*
document, [in] ULONG32 line, [in] ULONG32 column, [out, retval]
ISymUnmanagedMethod **pRetVal);
   HRESULT GetNamespaces([in] ULONG32 cNamespaces, [out] ULONG32
*pcNamespaces, [out] ISymUnmanagedNamespaces *namespaces[]);
   HRESULT GetSymAttribute([in] mdToken parent, [in] WCHAR *name, [in]
ULONG32 cBuffer, [out] ULONG32 *pcBuffer, [out] BYTE buffer[]);
   HRESULT GetSymbolStoreFileName([in] ULONG32 cchName, [out] ULONG
*pcchName, [out] WCHAR szName[]);
   HRESULT GetVariables([in] mdToken parent, [in] ULONG32 cVars, [out]
ULONG32 *pcVars, [out] ISymUnmanagedVariable *pVars[]);
   HRESULT GetUserEntryPoint([out, retval] mdMethodDef *pToken);
   HRESULT Initialize([in] IUnknown* importer, [in] const WCHAR*
fileName, [in] const WCHAR *searchPath, [in] IStream *pIStream);
HRESULT ReplaceSymbolStore([in] const WCHAR* filename, [in] IStream*
pIStream);
   HRESULT UpdateSymbolStore([in] const WCHAR* fileName, [in] Istream*
pIStream);
};
```

GetDocuments

```
HRESULT GetDocuments([in] ULONG32 cDocs, [out] ULONG32 *pcDocs, [out]
ISymUnmanagedDocument *pDocs[]);
```

Get the array of all the documents defined in the symbol store.

Parameter	Description
cDocs	The size of the array.
pcDocs	The number of documents available for return. No more than cDocs are actually returned in the array
pDocs	The array of documents

GetDocument

```
HRESULT GetDocument([in] WCHAR *url, [in] Guid language, [in] Guid
languageVendor, [in] Guid documentType, [out, retval]
ISymUnmanagedDocument** pRetVal);
```

Get a document specified by the language, vendor, and type.

Parameter	Description
url	The URL that identifies the document.
language	The document language. This argument can be specified as NULL.
languageVendor	The identity of the vendor for the document language. This argument can be specified as NULL.
documentType	The type of the document. This argument can be specified as NULL.
pRetVal	The specified document.

GetGlobalVariables

```
HRESULT GetGlobalVariables([in] ULONG32 cVars, [out] ULONG32 *pcVars,
[out] ISymUnmanagedVariable *pVars[]);
```

Get all global variables in the module.

Parameter	Description
cVars	The size of the variable array.
pcVars	The number of variables available for return. No more than cVars are actually returned in the array
pVars	The array of variables

GetMethod

```
HRESULT GetMethod([in] mdMethodDef token, [out, retval]
ISymUnmanagedMethod** pRetVal);
```

Gets a symbol reader method object given the token of a method

Parameter	Description
token	The Metadata token of the method.
pRetVal	The symbol reader method object for the given method

GetMethodByVersion

```
HRESULT GetMethodByVersion([in] mdMethodDef token, [in] int version,
[out, retval] ISymUnmanagedMethod** pRetVal);
```

Gets a symbol reader method object given the token of a method and its Edit and Continue version.

Parameter	Description

token	The Metadata token of the method.
version	The Edit and Continue version of the method.
pRetVal	The requested symbol reader method object.

GetMethodFromDocumentPosition

```
HRESULT GetMethodsFromDocumentPosition([in] ISymUnmanagedDocument*
document, [in] ULONG32 line, [in] ULONG32 column, [out, retval]
ISymUnmanagedMethod **pRetVal);
```

Given a position within a document, gets the symbol reader method object that contains the position.

Parameter	Description
document	The document in which the method is located.
line	The position of the line within the document. Lines are numbered beginning at 1.
column	The position of column within the document. Columns are numbered beginning at 1.
pRetVal	The reader method object for the specified position in the document.

GetNamespaces

```
HRESULT GetNamespaces([in] ULONG32 cNamespaces, [out] ULONG32
*pcNamespaces, [out] ISymUnmanagedNamespaces *namespaces[]);
```

Gets the namespaces defined at global scope within this symbol store.

Parameter	Description
cNamespaces	The size of the namespace array.
pcNamespaces	The number of namespaces available for return. No more than cNamespaces are actually returned in the array
namespaces	The array of namespaces

GetSymAttribute

```
HRESULT GetSymAttribute([in] mdToken parent, [in] WCHAR *name, [in]
ULONG32 cBuffer, [out] ULONG32 *pcBuffer, [out] BYTE buffer[]);
```

Gets a attribute value given the attribute name. This attribute is associated only with symbolic information and is not a Metadata custom attribute.

Parameter	Description
parent	The Metadata token for the object for which the attribute is

	requested.
name	The attribute name
cBuffer	The size of the buffer for the attribute value.
pcBuffer	The number of bytes available for return. No more than cBuffer are actually returned in the buffer
buffer	The buffer to receive the attribute value.

GetSymbolStoreFileName

```
HRESULT GetSymbolStoreFileName([in] ULONG32 cchName, [out] ULONG32
*pcchName, [out] WCHAR szName[]);
```

Gets the file name of the symbol store.

Parameter	Description
cchName	The size of the allocated array.
pcchName	The number of characters available for return. No more than cchName are actually returned in the array.
szName	The file name.

GetVariables

```
HRESULT GetVariables([in] mdToken parent, [in] ULONG32 cVars, [out]
ULONG32 *pcVars, [out] ISymUnmanagedVariable *pVars[]);
```

Return the non-local variables given the parent.

Parameter	Description
parent	The Metadata token for the type for which the variables are requested.
cVars	The size of the variable array.
pcVars	The number of variables available for return. No more than cVars are actually returned in the array
pVars	The array of variables.

GetUserEntryPoint

```
HRESULT GetUserEntryPoint([out, retval] mdMethodDef *pToken);
```

Get the Metadata token for the method that was specified as the user entry point for the module, if any. This would be, perhaps, the user's main method rather than compiler generated stubs before the main method.

Parameter	Description
pToken	The Metadata token for the method that is the user entry point for the module.

Initialize

```
HRESULT Initialize([in] IUnknown* importer, [in] const WCHAR* fileName,
[in] const WCHAR* searchPath, [in] IStream *pIStream);
```

Initialize the symbol reader with the metadata interface that this reader will be associated with, along with the file name of the module. This method can only be called once, and must be called before any other reader methods are called. If a stream is used, the filename and search path should be NULL.

Parameter	Description
importer	Pointer to the metadata importer interface that this reader will be associated with.
fileName	Pointer to the file name of the module.
searchPath	Pointer to the search path.
pIStream	Pointer to the stream that contains the symbol store.

ReplaceSymbolStore

```
HRESULT UpdateSymbolStore([in] const WCHAR* fileName, [in] IStream*
pIStream);
```

Updates the existing symbol store with a delta symbol store. This is much like UpdateSymbolStore, but the given delta acts as a complete replacement rather than an update.

Only one of the parameters, fileName or pIStream, needs to be specified. If fileName is specified, the symbol store will be updated with the symbols in that file. If pIStream is specified, the symbol store will be updated with the data from the IStream.

Parameter	Description
fileName	Pointer to the name of the file for the symbolic data.
pIStream	Pointer to the input stream for the symbolic data.

UpdateSymbolStore

```
HRESULT UpdateSymbolStore([in] const WCHAR* fileName, [in] IStream*
pIStream);
```

Updates the existing symbol store with a delta symbol store. This method is used in Edit and Continue scenarios as a way to update the symbol store to match deltas to the original PE file.

Only one of the parameters, fileName or pIStream, needs to be specified. If fileName is specified, the symbol store will be updated with the symbols in that file. If pIStream is specified, the symbol store will be updated with the data from the IStream.

Parameter	Description
fileName	Pointer to the name of the file for the symbolic data.

pIStream	Pointer to the input stream for the symbolic data.

3.1.7 ISymUnmanagedScope

The **ISymUnmanagedScope** interface represents a lexical scope within a **ISymUnmanagedMethod**. It provides access to the start and end offsets of the scope, as well as its child and parent scopes. The interface also provides access to all the locals defined within this scope.

IDL Declaration

```
interface ISymUnmanagedScope : IUnknown
{
   HRESULT GetChildren([in] ULONG32 cChildren, [out] ULONG32
*pcChildren, [out] ISymUnmanagedScope* children[]);
   HRESULT GetEndOffset([out, retval] ULONG32* pRetVal);
   HRESULT GetLocalCount([out, retval] ULONG32* pRetVal);
   HRESULT GetLocals([in] ULONG32 cLocals, [out] ULONG32 *pcLocals,
[out] ISymUnmanagedVariable* locals[]);
   HRESULT GetMethod([out, retval] ISymUnmanagedMethod** pRetVal);
   HRESULT GetNamespaces([in] ULONG32 cNamespaces, [out] ULONG32
*pcNamespaces, [out] ISymUnmanagedNamespaces *namespaces[]);
   HRESULT GetParent([out, retval] ISymUnmanagedScope** pRetVal);
   HRESULT GetStartOffset([out, retval] ULONG32* pRetVal);
};
```

GetChildren

```
HRESULT GetChildren([in] ULONG32 cChildren, [out] ULONG32 *pcChildren,
[out] ISymUnmanagedScope* children[]);
```

Gets the lexical scopes that are children of this lexical scope.

Parameter	Description
cChildren	The size of the allocated array.
pcChildren	The number of children available for return. No more than cChildren are actually returned in the array
children	Array of children.

GetEndOffset

```
HRESULT GetEndOffset([out, retval] ULONG32* pRetVal);
```

Gets the end byte offset of this lexical scope.

Parameter	Description
pRetVal	The end offset of this lexical scope.

GetLocalCount

```
HRESULT GetLocalsCounts([out, retval] ULONG32* pRetVal);
```

Get the count of local variables defined within this scope.

Parameter	Description
pRetVal	The number of local variables defined within this scope.

GetLocals

```
HRESULT GetLocals([in] ULONG32 cLocals, [out] ULONG32 *pcLocals, [out]
ISymUnmanagedVariable* locals[]);
```

Get the local variables within this lexical scope.

Parameter	Description
cLocals	The size of the local variable array.
pcLocals	The number of local variables available for return. No more than cLocals are actually returned in the array
locals	The array of local variables.

GetMethod

```
HRESULT GetMethod([out, retval] ISymUnmanagedMethod** pRetVal);
```

Gets the method that contains this lexical scope.

Parameter	Description
pRetVal	The method that contains this lexical scope.

GetNamespaces

```
HRESULT GetNamespaces([in] ULONG32 cNamespaces, [out] ULONG32
*pcNamespaces, [out] ISymUnmanagedNamespaces *namespaces[]);
```

Get the namespaces that are being used within this scope.

Parameter	Description
cNamespaces	The size of the namespace array.
pcNamespaces	The number of namespaces available for return. No more than cNamespaces are actually returned in the array
namespaces	The array of namespaces.

GetParent

```
HRESULT GetParent([out, retval] ISymUnmanagedScope** pRetVal);
```

Gets the parent lexical scope of this scope.

Parameter	Description
pRetVal	The parent lexical scope of this scope.

GetStartOffset

```
HRESULT GetStartOffset([out, retval] ULONG32* pRetVal);
```

Gets the start byte offset of this lexical scope.

Parameter	Description
pRetVal	The start byte offset of this lexical scope.

3.1.8 ISymUnmanagedVariable

The **ISymUnmanagedVariable** interface represents a symbol within a symbol store. This could a parameter, a local variable, or a field.

IDL Declaration

```
interface ISymUnmanagedVariable : IUnknown
{
   HRESULT GetAddressField1([out, retval] ULONG32* pRetVal);
   HRESULT GetAddressField2([out, retval] ULONG32* pRetVal);
   HRESULT GetAddressField3([out, retval] ULONG32* pRetVal);
   HRESULT GetAddressKind([out, retval] ULONG32* pRetVal);
   HRESULT GetAttributes([out, retval] ULONG32* pRetVal);
   HRESULT GetEndOffset([out, retval] ULONG32* pRetVal);
   HRESULT GetName([in] ULONG32 cchName, [out] ULONG32 *pcchName, [out]
WCHAR szName[]);
   HRESULT GetSignature([in] ULONG32 cSig, [out] ULONG32 *pcSig, [out]
BYTE sig[]);
   HRESULT GetStartOffset([out, retval] ULONG32* pRetVal);
};
```

GetAddressField1

```
HRESULT GetAddressField1([out, retval] ULONG32* pRetVal);
```

Gets the first address of the variable.

Parameter	Description
pRetVal	The first address of the variable.

GetAddressField2

```
HRESULT GetAddressField2([out, retval] ULONG32* pRetVal);
```

Gets the second address of the variable.

Parameter	Description
pRetVal	The second address of the variable.

GetAddressField3

```
HRESULT GetAddressField3([out, retval] ULONG32* pRetVal);
```

Gets the third address of the variable.

Parameter	Description
pRetVal	The third address of the variable.

GetAddressKind

```
HRESULT GetAddressKind([out, retval] ULONG32* pRetVal);
```

Gets the types of the addresses.

Parameter	Description
pRetVal	The types of the addresses.

GetAttributes

```
HRESULT GetAttributes([out, retval] ULONG32* pRetVal);
```

Gets the variable attributes.

Parameter	Description
pRetVal	The variable attributes.

GetEndOffset

```
HRESULT GetEndOffset([out, retval] ULONG32* pRetVal);
```

Gets the end offset of this variable within its parent. If this is a local variable within a scope, this will fall within the offsets defined for the scope.

Parameter	Description
pRetVal	The end offset of the variable.

GetName

```
HRESULT GetName([in] ULONG32 cchName, [out] ULONG32 *pcchName, [out]
WCHAR szName[]);
```

Gets the name of the variable.

Parameter	Description
cchName	The size of the allocated character array.
pcchName	The number of characters available for return. No more than cchName are actually returned in the array
szName	The array of characters that represent the name.

GetSignature

```
HRESULT GetSignature([in] ULONG32 cSig, [out] ULONG32 *pcSig, [out]
BYTE sig[]);
```

Gets the signature of this variable as an opaque byte array.

Parameter	Description
cSig	The size of the allocated byte blob.
pcSig	The number of bytes available for return. No more than cSig are actually returned in the array
sig	The signature byte array.

GetStartOffset

```
HRESULT GetStartOffset([out, retval] ULONG32* pRetVal);
```

Gets the start offset of this variable within its parent. If this is a local variable within a scope, this will fall within the offsets defined for the scope.

Parameter	Description
pRetVal	The start byte offset of the variable.

3.1.9 ISymUnmanagedWriter

The ISymUnmanagedWriter interface represents a symbol writer for managed code. The interface provides methods to define documents, sequence points, lexical scopes, and variables. The class **CoSymWriter** is the Microsoft implementation of this class that targets the PDB symbolic information format.

IDL Declaration

```
interface ISymUnmanagedWriter : IUnknown
{
   HRESULT CloseMethod();
   HRESULT CloseScope([in] ULONG32 endOffset);
   HRESULT Close();
```

```
   HRESULT DefineDocument([in] const WCHAR *url, [in] const Guid*
language, [in] const Guid* languageVendor, [in] const Guid*
documentType, [out, retval] ISymUnmanagedDocumentWriter** pRetVal);
   HRESULT DefineField([in] mdTypeDef parent, [in] const WCHAR *name,
[in] ULONG32 attributes, ULONG32 cSig, [in] unsigned char signature[],
[in] ULONG32 addressKind, [in] ULONG32 addr1, [in] ULONG32 addr2, [in]
ULONG32 addr3);
   HRESULT DefineGlobalVariable([in] const WCHAR *name, [in] ULONG32
attributes, [in] ULONG32 cSig, [in] unsigned char signature[], [in]
ULONG32 addressKind, [in] ULONG32 addr1, [in] ULONG32 addr2, [in]
ULONG32 addr3);
   HRESULT DefineLocalVariable([in] const WCHAR *name, [in] ULONG32
attributes, [in] ULONG32 cSig, [in] unsigned char signature[], [in]
ULONG32 addressKind, [in] ULONG32 addr1, [in] ULONG32 addr2, [in]
ULONG32 addr3, [in] ULONG32 startOffset, [in] ULONG32 endOffset);
   HRESULT DefineParameter([in] const WCHAR *name, [in] ULONG32
attributes, [in] ULONG32 sequence, [in] ULONG32 addressKind, [in]
ULONG32 addr1, [in] ULONG32 addr2, [in] ULONG32 addr3);
   HRESULT DefineSequencePoints([in] ISymUnmanagedDocumentWriter*
document, [in] ULONG32 spCount, [in] ULONG32 offsets[], [in] ULONG32
startLines[], [in] ULONG32 startColumns[], [in] ULONG32 endLines[],
[in] ULONG32 endColumns[]);
   HRESULT GetDebugInfo([in] IMAGE_DEBUG_DIRECTORY *pIDD, [in] ULONG32
cData, [out] ULONG32 *pcData, [out] BYTE data[]);
   HRESULT Initialize([in] IUnknown* emitter, [in] const WCHAR*
fileName, [in] IStream* pIStream, [in] BOOL fFullBuild);
   HRESULT OpenMethod([in] mdMethodDef method);
   HRESULT OpenNamespace([in] const WCHAR *name);
   HRESULT OpenScope([in] ULONG32 startOffset, [out, retval] ULONG32*
pRetVal);
   HRESULT RemapToken([in] mdToken oldToken, [in] mdToken newToken);
   HRESULT SetSymAttribute([in] mdToken parent, [in] const WCHAR *name,
[in] ULONG32 cData, [in] unsigned char data[]);
   HRESULT SetScopeRange([in] ULONG32 scopeId, [in] ULONG32
startOffset, [in] ULONG32 endOffset);
   HRESULT SetUserEntryPoint([in] mdMethodDef entryMethod);
   HRESULT UsingNamespace([in] const WCHAR *fullName);
};
```

CloseMethod

```
HRESULT CloseMethod();
```

Closes the current method. Once a method is closed, no more symbols can be defined within it.

CloseNamespace

```
HRESULT CloseNamespace();
```

Closes the most recently opened namespace.

CloseScope

```
HRESULT CloseScope([in] ULONG32 endOffset);
```

Closes the current lexical scope. Once a scope is closed no more variables can be defined within it.

Parameter	Description
endOffset	Points past the last instruction in the scope.

Close

```
HRESULT Close();
```

Closes **ISymUnmanagedWriter** and commits the symbols to the symbol store. **ISymUnmanagedWriter** becomes invalid after this call for further updates.

DefineDocument

```
HRESULT DefineDocument([in] const WCHAR *url, [in] const Guid*
language, [in] const Guid* languageVendor, [in] const Guid*
documentType, [out, retval] ISymUnmanagedDocumentWriter** pRetVal);
```

Defines a source document.

Parameter	Description
url	The URL that identifies the document.
language	The document language. This argument can be NULL.
languageVendor	The identity of the vendor for the document language. This argument can be NULL.
documentType	The type of the document. This argument can be NULL.
pRetVal	Pointer to a ISymUnmanagedDocumentWriter object that represents the object.

DefineField

```
HRESULT DefineField([in] mdTypeDef parent, [in] const WCHAR *name, [in]
ULONG32 attributes, ULONG32 cSig, [in] unsigned char signature[], [in]
ULONG32 addressKind, [in] ULONG32 addr1, [in] ULONG32 addr2, [in]
ULONG32 addr3);
```

Defines a field in a type or a global field.

Parameter	Description
parent	The Metadata type token.
name	The field name.
attributes	The field attributes.
cSig	The size of the signature.
signature	The field signature.
addressKind	The address types for addr1 and addr2.
addr1	First address for the field specification.
addr2	Second address for the field specification.
addr3	Third address for the field specification.

DefineGlobalVariable

```
HRESULT DefineGlobalVariable([in] const WCHAR *name, [in] ULONG32
attributes, [in] ULONG32 cSig, [in] unsigned char signature[], [in]
ULONG32 addressKind, [in] ULONG32 addr1, [in] ULONG32 addr2, [in]
ULONG32 addr3);
```

Defines a single global variable.

Parameter	Description
name	The global variable name.
attributes	The global variable attributes.
cSig	The size of the signature.
signature	The global variable signature.
addressKind	The address types for addr1, addr2, and addr3 defined using **CorSymAddrKind**.
addr1	First address for the global variable specification.
addr2	Second address for the global variable specification.
addr3	Third address for the global variable specification.

DefineLocalVariable

```
HRESULT DefineLocalVariable([in] const WCHAR *name, [in] ULONG32
attributes, [in] ULONG32 cSig, [in] unsigned char signature[], [in]
ULONG32 addressKind, [in] ULONG32 addr1, [in] ULONG32 addr2, [in]
ULONG32 addr3, [in] ULONG32 startOffset, [in] ULONG32 endOffset);
```

Defines a local variable in the current lexical scope. This method can be called multiple times for a variable of the same name that has multiple homes throughout a scope. Note that start and end offsets must not overlap in such a case.

Parameter	Description
name	The local variable name.
attributes	The local variable attributes.
cSig	The size of the signature.
signature	The local variable signature.
addressKind	The address types for addr1, addr2, and addr3 defined using **CorSymAddrKind**.
addr1	First address for the local variable specification.
addr2	Second address for the local variable specification.
addr3	Third address for the local variable specification.
startOffset	If zero, it is ignored and the variable is defined over the entire scope. If non-zero, it must fall with the extent (offsets) of the current scope.
endOffset	If zero, it is ignored and the variable is defined over the entire scope. If non-zero, it must fall with the extent (offsets) of the current scope.

DefineParameter

```
HRESULT DefineParameter([in] const WCHAR *name, [in] ULONG32
attributes, [in] ULONG32 sequence, [in] ULONG32 addressKind, [in]
ULONG32 addr1, [in] ULONG32 addr2, [in] ULONG32 addr3);
```

Defines a parameter.

Parameter	Description
name	The parameter name.
attributes	The parameter attributes.
sequence	The parameter sequence number
addressKind	The address types for addr1, addr2 and addr3.
addr1	First address for the parameter specification.
addr2	Second address for the parameter specification.
addr3	Third address for the parameter specification.

DefineSequencePoints

```
HRESULT DefineSequencePoints([in] ISymUnmanagedDocumentWriter*
document, [in] ULONG32 spCount, [in] ULONG32 offsets[],[in] ULONG32
startLines[], [in] ULONG32 startColumns[], [in] ULONG32 endLines[],
[in] ULONG32 endColumns[]);
```

Define a group of sequence points within the current method. Each line and each column defines the start of a statement within a method. The arrays should be

sorted in the increasing order of offsets. The offset is always the offset from the start of the method in bytes.

Parameter	Description
document	The document object for which the sequence points are being defined.
spCount	The number of sequence points which is also the size of the arrays, offsets, lines, and columns.
offsets	The sequence point offsets measured from the beginning of methods.
startLines	The start lines for the sequence points.
startColumns	The start positions for the sequence points.
endLines	The end lines for the sequence points.
endColumns	The end positions for the sequence points.

GetDebugInfo

```
HRESULT GetDebugCVInfo([in] IMAGE_DEBUG_DIRECTORY *pIDD, [in] ULONG32
cData, [out] ULONG32* pcData, [out] BYTE data[]);
```

Returns a blob of data that a compiler or other PE file writer must emit into the PE header. If a symbol store does not require any such information in the PE header, this method will return a data size of zero. The IMAGE_DEBUG_DIRECTORY will be filled in by the symbol writer, except for the offset and rva fields. The program emitting the PE file should emit a debug directory entry with pIDD as an entry, then emit the blob and ensure that the offset and rva fields point to the blob.

Parameter	Description
pIDD	Pointer to the IMAGE_DEBUG_DIRECTORY that the writer will fill in.
cData	The size of the allocated byte blob.
pcData	The number of bytes available for return. No more than cData are actually returned in the array
data	The bytes in the blob of data.

Initialize

```
HRESULT Initialize([in] IUnknown* emitter, [in] const WCHAR* fileName,
[in] IStream* pIStream, [in] BOOL fFullBuild);
```

Sets the metadata emitter interface that this writer will be associated with. Also sets the output file name where the debugging symbols will be written. This method can be called only once and must be called before any other writer methods are called.

Parameter	Description
emitter	Pointer to the Metadata emitter interface.
fileName	Pointer to a file name for which the debugging symbols will be

	written. Some writers may require a file name, while others may not. If a file name is specified for a writer that does not use file names, this parameter will be ignored.
pIStream	Pointer to an input stream. If this optional parameter is specified, the symbol writer will emit symbols to the given stream.
fFullBuild	TRUE indicates that this is a full rebuild. FALSE indicates an incrememtal compilation.

OpenMethod

```
HRESULT OpenMethod([in] mdMethodDef method);
```

Opens a method to emit symbol information into. The given method becomes the current method for calls to define sequence pints, parameters, and lexical scopes. There is an implicit lexical scope around the entire method. Reopening a method that has been previously closed effectively erases any previously defined symbols for the method.

Parameter	Description
method	The Metadata token for the method to be opened.

OpenScope

```
HRESULT OpenScope([in] ULONG32 startOffset, [out, retval] ULONG32*
pRetVal);
```

Opens a new lexical scope in the current method. The scope becomes the new current scope and is effectively pushed onto a stack of scopes. Scopes must form a hierarchy. Siblings are not allowed to overlap.

Parameter	Description
startOffset	The offset, in bytes from the beginning of the method of the first instruction in the lexical scope.
pRetVal	An opaque scope id that can be used with SetScopeRange to define a scope's start and end offsets at a later time. In this case, the offsets passed to OpenScope and CloseScope are ignored. A scope id is only valid in the current method.

OpenNamespace

```
HRESULT OpenNamespace([in] const WCHAR *name);
```

Opens a new new namespace. Call this before defining methods or variables that live within a namespace. Namespaces can be nested.

Parameter	Description
name	The name of the new namespace.

RemapToken

```
HRESULT RemapToken([in] mdToken oldToken, [in] mdToken newToken);
```

Tells the symbol writer that a Metadata token has been remapped as the Metadata was emitted. If the symbol writer has stored the old token within the symbol store, it must either update the stored token to the new value, or persist the map for the corresponding symbol reader to remap during the read phase.

Parameter	Description
oldToken	The original Metadata token.
newToken	The remapped Metadata token.

SetSymAttribute

```
HRESULT SetSymAttribute([in] mdToken parent, [in] const WCHAR *name,
[in] ULONG32 cData, [in] unsigned char data[]);
```

Defines a attribute given the attribute name and the attribute value. This attribute is associated only with symbolic information and is not a Metadata custom attribute.

Parameter	Description
parent	The Metadata token for which the attribute is being defined.
name	The attribute name.
cData	The size of the attribute value.
data	The attribute value.

SetScopeRange

```
HRESULT SetScopeRange([in] ULONG32 scopeId, [in] ULONG32 startOffset,
[in] ULONG32 endOffset);
```

Defines the offset range for the given lexical scope.

Parameter	Description
scopeId	The id of the lexical scope.
startOffset	The byte offset of the beginning of the lexical scope.
endOffset	The byte offset of the end of the lexical scope.

SetUserEntryPoint

```
HRESULT SetUserEntryPoint([in] mdMethodDef entryMethod);
```

Identifies the method that the user has defined as the entry point for this module. This would be, perhaps, the user's main method rather than the compiler generated stubs before main.

Parameter	Description
entryMethod	The Metadata token for the method that is the user entry point.

UsingNamespace

```
HRESULT UsingNamespace([in] const WCHAR *fullName);
```

Specifies that the given, fully qualified namespace name is being used within the currently open lexical scope. Closing the current scope will also stop using the namespace, and the namespace will be in use in all scopes that inherit from the currently open scope.

Parameter	Description
fullName	The fully qualified namespace name.

4 Typedefs

4.1.1 CorSymAddrKind

This enumeration is used to specify address types for local variables, parameters, and fields in the methods **DefineLocalVariable**, **DefineParameter**, and **DefineField**. addr1 and addr2 refer to the address parameters in those methods.

See also: DefineLocalVariable, DefineParameter, DefineField, DefineGlobalVariable.

IDL Declaration

```
typedef enum CorSymAddrKind
{
    ADDR_BITFIELD = 9;
    ADDR_IL_OFFSET = 1;
    ADDR_NATIVE_OFFSET = 5;
    ADDR_NATIVE_REGISTER = 3;
    ADDR_NATIVE_REGREG = 6;
    ADDR_NATIVE_REGREL = 4;
    ADDR_NATIVE_REGSTK = 7;
    ADDR_NATIVE_RVA = 2;
    ADDR_NATIVE_STKREG = 8;
} CorSymAddrKind;
```

ADDR_BITFIELD

```
ADDR_BITFIELD = 9;
```

Specifies a bit field. addr1 is the position where the field starts. addr2 is the field length.

ADDR_IL_OFFSET

```
ADDR_IL_OFFSET = 1;
```

Specifies an IL offset. addr1 is the IL local variable or parameter index.

ADDR_NATIVE_OFFSET

```
ADDR_NATIVE_OFFSET = 5;
```

Specifies a native offset. addr1 is the offset from start of the parent.

ADDR_NATIVE_REGISTER

```
ADDR_NATIVE_REGISTER = 3;
```

Specifies a native register address. addr1 is the register in which the variable is stored.

ADDR_NATIVE_REGREG

```
ADDR_NATIVE_REGREG = 6;
```

Specifies a register relative address. addr1 is the register low. addr2 is the register high.

ADDR_NATIVE_REGREL

```
ADDR_NATIVE_REGREL = 4;
```

Specifies a register relative address. addr1 is the register. addr2 is the offset.

ADDR_NATIVE_REGSTK

```
ADDR_NATIVE_REGSTK = 47;
```

Specifies a register relative address. addr1 is the register low. addr2 is the register stack, addr3 is the offset.

ADDR_NATIVE_RVA

```
ADDR_NATIVE_RVA = 2;
```

Specifies a native RVA. addr1 is the RVA in the module.

ADDR_NATIVE_STKREG

```
ADDR_NATIVE_STKREG = 48;
```

Specifies a register relative address. addr1 is the register stack. addr2 is the offset, addr3 is the register high.