Reference Guide to the Framework Presented in
“Implementing Reusable Solvers: An
objected-Oriented Framework for Operations
Research Algorithms
by
J. Ruark

OR 328-98     June 1998

Working paper OR 328-98, MIT Operations Research Center

John D. Ruark

ruark@post.harvard.edu
Massachusetts Institute of Technology
77 Massachusetts Ave., E40-139, Cambridge, MA 02139

This document is the reference guide to the framework proposed and described in the doctoral thesis "Implementing Reusable Solvers: An Object-Oriented Framework for Operations Research Algorithms" by John Ruark [1]. Copies of the thesis may be obtained through the MIT Libraries' Document Services (77 Massachusetts Ave., 14-0551, Cambridge, MA 02139, 617-253-5668, docs@mit.edu).

To maintain consistency of presentation, some passages of this text are based closely on Microsoft Developer Network Library and Microsoft Windows Platform SDK documentation files. Such paraphrasing or quotations of common idioms, to ensure consistency wherever they occur, are hereby acknowledged as deriving from these sources.

Reference


Copyright © 1998, John D. Ruark. All rights reserved.
The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this reference in whole or in part.
# Table of Contents

1 Interfaces

1.1 IEnumRDESCPROPIDINFO ................................................................. 5
1.2 IEnumRDESCRIPTIONPROPERTY .................................................. 5
1.3 IEnumRREGSOLVERINFO .............................................................. 6
1.4 IRDataAdvise ........................................................ .......................... 7
1.5 IRDataAdviseHolder ................................................................. 8
1.6 IRDataElement .......................................................... ........................ 10
1.7 IRDataElement1 .......................................................... ..................... 16
1.8 IRDataElement2 .......................................................... ..................... 18
1.9 IRDataElementAccess .......................................................... ............... 23
1.10 IRDataElementClone .......................................................... ............... 25
1.11 IRDataElementCreator .......................................................... ............... 26
1.12 IRDataElementItemProperties .................................................. 28
1.13 IRDataElementItemPropertiesChange .................................... 31
1.14 IRDataElementScalar .......................................................... .......... 33
1.15 IRDataSource .......................................................... ........................ 34
1.16 IRDescriptionPropEnumeration ............................................ 42
1.17 IRDescriptionPropRegistration ............................................ 45
1.18 IRDimension .......................................................... ........................ 47
1.19 IRDimensionClone .......................................................... ............... 49
1.20 IRDimensionCreator .......................................................... ............... 50
1.21 IRMappingAdvise .......................................................... ............... 53
1.22 IRSet .......................................................... ............................ 54
1.23 IRSetClone .......................................................... ........................ 60
1.24 IRSetCreator .......................................................... ........................ 61
1.25 IRSetModifier .......................................................... ........................ 62
1.26 IRSolver .......................................................... ............................ 67
1.27 IRSolverAdvise .......................................................... ............... 73
2 Global library functions ........................................ 157
  2.1 RLoadRegSolverInfo ........................................ 157
  2.2 RLoadSolverInfo ........................................... 158
  2.3 RLoadSolverInfoClsid ....................................... 159
2.4 RLookupInfoGuid ................................................................. 160

3   Structures ......................................................................................... 161
    3.1 RDESCPROPIDINFO ................................................................. 161
    3.2 RDESCRIPTIONPROPERTY .................................................. 161
    3.3 RREGSOLVERINFO .............................................................. 162

4   Enumerations .................................................................................... 163
    4.1 RSD_DESCRIPTIONFLAGS .................................................. 163
    4.2 RSI_SOLVERFLAGS ............................................................... 163
    4.3 RSOLVE_DATA_ACCESS .......................................................... 164
    4.4 RSOLVE_MAPPING_PREFERENCES ........................................... 164
    4.5 RSOLVE_SOLVE_COMPLETE_RESULTS ...................................... 165
    4.6 RSOLVE_SOLVE_NOTIFY_ACTIONS ......................................... 165
    4.7 RSOLVE_SOLVE_NOTIFY_FLAGS .............................................. 166
    4.8 RSOLVE_SOLVER_ASSIGNMENT_FLAGS .................................. 166
    4.9 RSOLVE_SOLVER_INFO_FLAGS ............................................... 167
    4.10 RSOLVE_SOLVER_INPUT_CHANGE_FLAGS ................................ 167
    4.11 RSOLVE_SOLVER_OUTPUT_CHANGE_FLAGS ............................... 167
    4.12 RSOLVE_SOLVER_STATUS ..................................................... 168
    4.13 RSOLVE_SOLVERFACTORY_CREATEFLAGS ............................... 168
    4.14 RSOLVE_SOLVERFACTORYDESTROYFLAGS ............................... 169

5   Constants .......................................................................................... 170
    5.1 Interface identifiers (IIDs) ..................................................... 170
    5.2 Class identifiers (CLSIDs) ..................................................... 171
    5.3 Other GUIDs ............................................................................. 172
    5.4 Error codes (HRESULTs) ........................................................... 172
This section describes the interfaces defined by the framework.

1.1 IEnumRDESCPROMPIDINFO

The IEnumRDESCPROMPIDINFO interface is used to enumerate an array of RDESCPROMPIDINFO structures. IEnumRDESCPROMPIDINFO has the same methods as all enumerator interfaces: Next, Skip, Reset, and Clone. For general information on these methods, refer to IEnumXXXX in the COM documentation.

When to implement. IEnumRDESCPROMPIDINFO must be implemented by all description property managers to support calls to IRDescriptionPropEnumeration::EnumRDESCPROMPIDINFO. Hence, it is only necessary to implement this interface when implementing a custom description property manager.

When to use. Call the methods of IEnumRDESCPROMPIDINFO to enumerate the description property identifiers of the description properties registered with the core services.

Methods in vtable order

**IUnknown methods:**
- **QueryInterface** Returns pointers to supported interfaces.
- **AddRef** Increments reference count.
- **Release** Decrements reference count.

**IEnumRDESCPROMPIDINFO methods:**
- **Next** Advances to the next element
- **Skip** Skips elements
- **Reset** Resets to the start of the list
- **Clone** Creates a new enumerator at the same place

The prototypes of the methods are as follows:

- `HRESULT Next(ULONG celt, RDESCPROMPIDINFO * rgelt, ULONG * pceltFetched)`
- `HRESULT Skip(ULONG celt)`
- `HRESULT Reset(void)`
- `HRESULT Clone(IEnumRDESCPROMPIDINFO ** ppenum)`

1.2 IEnumRDESCRIPTIONPROPERTY

The IEnumRDESCRIPTIONPROPERTY interface is used to enumerate an array of RDESCRIPTIONPROPERTY structures. IEnumRDESCRIPTIONPROPERTY has the same methods as all enumerator interfaces: Next, Skip, Reset, and Clone. For general information on these methods, refer to IEnumXXXX in the COM documentation.
**IEnumRREGSOLVERINFO**

**When to implement.** `IEnumRDESCRIPTIONPROPERTY` must be implemented by all solver description objects to support calls to `IRSolverDescriptionProperties::EnumDescriptionProperties`. The generic implementation of SolverInfo provides an implementation of this interface. Hence, it is only necessary to implement this interface when implementing a custom solver description object.

**When to use.** Call the methods of `IEnumRDESCRIPTIONPROPERTY` to enumerate the description properties exposed by a solver.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IEnumRDESCRIPTIONPROPERTY methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next</td>
</tr>
<tr>
<td>Skip</td>
</tr>
<tr>
<td>Reset</td>
</tr>
<tr>
<td>Clone</td>
</tr>
</tbody>
</table>

The prototypes of the methods are as follows:

```c
HRESULT Next(ULONG celt, RDESCRIPTIONPROPERTY * rgelt, ULONG * pceltFetched)
HRESULT Skip(ULONG celt)
HRESULT Reset(void)
HRESULT Clone(IEnumRDESCRIPTIONPROPERTY ** ppenum)
```

1.3 **IEnumRREGSOLVERINFO**

The `IEnumRREGSOLVERINFO` interface is used to enumerate an array of `RREGSOLVERINFO` structures. `IEnumRREGSOLVERINFO` has the same methods as all enumerator interfaces: `Next`, `Skip`, `Reset`, and `Clone`. For general information on these methods, refer to `IEnumXXXX` in the COM documentation.

**When to implement.** `IEnumRREGSOLVERINFO` must be implemented by all solver registrars to support calls to `IRSolverEnumeration::EnumSolversWithInfo`. Hence, it is only necessary to implement this interface when implementing a custom solver registrar.

**When to use.** Call the methods of `IEnumRREGSOLVERINFO` to enumerate the solvers that are registered with the core services.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IEnumRREGSOLVERINFO methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next</td>
</tr>
<tr>
<td>Skip</td>
</tr>
</tbody>
</table>
IRDataAdvise

Reset
Clones to the start of the list

The prototypes of the methods are as follows:

- HRESULT Next(ULONG celt, RDESCPROPIDINFO * rgelt, ULONG * pceltFetched)
- HRESULT Skip(ULONG celt)
- HRESULT Reset(void)
- HRESULT Clone(IEnumRREGSOLVERINFO ** ppenum)

1.4 IRDataAdvise

The IRDataAdvise interface enables solver sites, mappings, clients, and other interested objects to receive data change notifications from a data source. For an advisory connection to exist, the object that is to receive notifications must implement or contain a sub-object that implements IRDataAdvise. Mappings use IRDataAdvise to be notified when the data they will map is ready.

Objects interested in receiving the data change notifications should establish an advisory connection with the data source by calling IRDataSource::Advise. A connection can be terminated by calling IRDataSource::Unadvise. While the connection is active, the data source may call the IRDataAdvise::OnDataChange notification at any time.

When to implement. IRDataAdvise should be implemented on those objects that are interested in receiving directly data change notifications.

When to use. Data sources use IRDataAdvise to notify their advisory connections that data has changed. Normally, other objects do not need to use this interface.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRDataAdvise method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDataChange</td>
<td>The data source has changed.</td>
</tr>
</tbody>
</table>

1.4.1 IRDataAdvise::OnDataChange

Notifies the object receiving advisory notifications that an element managed by the data source has changed.

- HRESULT OnDataChange();

Return values

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The message was processed.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
IRDataAdviseHolder

Because these are notifications, the object calling the OnDataChange method can ignore the return value.

1.5 IRDataAdviseHolder

The IRDataAdviseHolder interface contains methods that create and manage advisory connections between a data source and one or more advise sinks. Its methods are intended to be used to implement the advisory methods of IRDataSource. IRDataAdviseHolder is implemented on an advise holder object. Its methods establish and delete data source advisory connections and send notification of data change of data elements managed by the data source to an object that requires this notification, such as a mapping, which must contain an advise sink.

Advise sinks are objects that require notification of data change and implement the IRDataAdvise interface.

When to implement. Typically, use the framework-provided implementation of IRDataAdviseHolder to simplify the implementation of the Advise and Unadvise methods in the IRDataSource interface, and to send notifications as appropriate. It would be necessary to implement IRDataAdviseHolder only in the case where there may be a need for a custom data source advise holder object, whose methods are to be used to implement the IRDataSource methods in a set of servers.

When to use. The implementation of the advisory methods of IRDataSource can call the methods in IRDataAdviseHolder. The first time the data source receives a call to IRDataSource::Advise, create an object of CLSID CLSID_RDataAdviseHolder to create an instance of the framework-provided advise holder and get a pointer to its IRDataAdviseHolder interface. Then, in implementing the IRDataSource interface on the data object, delegate implementations of the Advise and Unadvise methods to the corresponding methods in IRDataAdviseHolder.

When interesting data changes occur, call IRDataAdviseHolder::SendOnDataChange from the data source to carry out the necessary notifications.

Methods in vtable order

IUnknown methods: Description
QueryInterface Returns pointers to supported interfaces.
AddRef Increments reference count.
Release Decrements reference count.
IRDataAdviseHolder methods:
Advise Establishes advisory connection with sink.
Unadvise Destroys advisory connection with sink.
SendOnDataChange Advises sinks of data change.

1.5.1 IRDataAdviseHolder::Advise

Creates an advisory connection between an advise sink and a data source for receiving notifications. This is typically called in the implementation of IRDataSource::Advise.
HRESULT Advise(
    [in] LPRDATASOURCE pSource,
    [in] LPRDATAADVISE pAdvise,
    [out] DWORD* pdwCookie
);  

Parameters

pSource

[in] Pointer to the data source that will provide notifications.

pAdvise

[in] Pointer to the advise sink that will receive notifications.

pdwCookie

[out] Pointer to a DWORD that upon return will contain a cookie that identifies this advisory connection. The client can later destroy this connection by passing this cookie to IRDataAdviseHolder::Unadvise. If this value is zero, the connection was not established.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The connection was established.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_INVALIDARG</td>
<td>Either the data source or advise sink pointer was invalid.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The DWORD cookie pointer was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The data source typically calls this method in the IRDataSource::Advise method, to manage the list of advise connections.

1.5.2 IRDataAdviseHolder::SendOnDataChange

Sends notifications to each advise sink for which there is a connection established by calling the IRDataAdvise::OnDataChange method for each advise sink currently being handled by this instance of the data advise holder object.

HRESULT SendOnDataChange();
IRDataElement

Return Values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The call to IDataAdvise::OnDataChange was made.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>

Remarks

The data source typically calls this method when it detects a change that would be of interest to an advise sink that has previously requested notification.

1.5.3 IRDataAdviseHolder::Unadvise

Deletes an existing advisory connection between an advise sink and a data source that had previously been created by a call to IRDataAdviseHolder::Advise. This is typically called in the implementation of IRDataSource::Unadvise.

```c
HRESULT Unadvise(
    [in] DWORD dwCookie
);
```

Parameter

dwCookie

[in] A DWORD cookie that identifies the connection to delete. This cookie must have been obtained from a prior call to IRDataAdviseHolder::Advise.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The specified connection was successfully deleted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_E_NOCONNECTION</td>
<td>The specified dwCookie is not a valid connection.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.6 IRDataElement

The interface IRDataElement is the fundamental interface of data elements in the framework. It provides the basic functionality required of all data elements, including dimension access, item access and modification, and access to the data source that manages the data element.

When to implement. Implement this interface on all data element objects. In particular, output data elements of solvers must implement this interface.
**When to use.** Use this interface to access the fundamental characteristics of a data element.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRDataElement methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetName</td>
<td>Returns name of the data element.</td>
</tr>
<tr>
<td>GetDimensionCount</td>
<td>Returns number of dimensions.</td>
</tr>
<tr>
<td>GetDimension</td>
<td>Returns a dimension.</td>
</tr>
<tr>
<td>GetAt</td>
<td>Returns a value within the element.</td>
</tr>
<tr>
<td>SetAt</td>
<td>Sets a value within the element.</td>
</tr>
</tbody>
</table>
| GetDataSource         | Returns the data source that manages this element.

1.6.1 IRDataElement::GetAt

Retrieves a value within the data element.

```c
HRESULT GetAt(
    [in] long cDim,
    [in, size_is(cDim)] long rgDim[],
    [out, retval] RSOLVER_DATA* pData
);
```

**Parameters**

- **cDim**
  
  [in] Number of dimensions in the data element.

- **rgDim[]**
  
  [in] An array containing the index of each dimension that identifies which value to retrieve within the data element. Element \( i \) in the array is the index for dimension number \( i \). This array has a size equal to the \( cDim \) parameter.

- **pData**
  
  [in] Pointer to a caller-allocated RSOLVER_DATA structure that upon successful completion contains the value of the item to retrieve within the data element.

**Return values**

<table>
<thead>
<tr>
<th>RSOLVE_E_INVALIDINDEX</th>
<th>The item was successfully retrieved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>One of the indices in the rgDim array was invalid.</td>
</tr>
</tbody>
</table>
IRDataElement

<table>
<thead>
<tr>
<th>E_OUTOFMEMORY</th>
<th>The operation failed due to insufficient memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_INVALIDARG</td>
<td>One of the input parameters was invalid.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output pointer was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The valid indices are specified by the valid indices for the dimension associated with the data element vector. See IRDimension for more details.

1.6.2 IRDataElement::GetDataSource

Returns the data source that manages this data element, and the index number within the data source that identifies this data element.

HRESULT GetDataSource(  
    [in] REFIID riid,  
    [out, iid_is(riid)] LPVOID* ppVoid,  
    [out] UINT* puElement  
);

Parameters

riid

[in] Identifier of the interface of the data source to return. Normally this is IID_IRDataSource.

ppVoid

[out] Pointer to location that upon successful completion contains the requested interface pointer.

puElement

[out] Pointer to a UINT that upon successful completion contains the index of the data element within the data source.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOINTERFACE</td>
<td>The data source does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>
A data source can manage multiple data elements. Thus, a data element is identified within a data source by an index, which is returned in the last parameter of this method. To recover the data element from the outputs of this method, call `IRDataSource::GetDataAccessor` on the returned interface pointer, passing it the index returned in the last parameter.

### 1.6.3 IRDataElement::GetDimension

Returns a dimension object that characterizes one of the dimensions of the data element.

```cpp
HRESULT GetDimension(
    [in] UINT uDim,
    [out, retval] LPRDIMENSION* ppDim
);
```

**Parameters**

- **uDim**
  
  `[in]` The index of the dimension to retrieve. This value should be less than the number returned from `GetDimensionCount`.

- **ppDim**
  
  `[out]` Pointer to location that upon successful completion contains the `IRDimension` interface pointer of the requested dimension object. Upon failure, `*ppDim` is set to zero.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The parameter <code>uDim</code> was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output pointer was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

### 1.6.4 IRDataElement::GetDimensionCount

Returns the number of dimensions associated with this data element.
IRDataElement

HRESULT GetDimensionCount(
    [out, retval] UINT* pDims
);

Parameter

pDims

[out] Pointer to a UINT that upon successful completion contains the number of dimensions associated with the data element. Upon failure, *pDims is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

Scalar values should set *pDims to zero, not return an error.

1.6.5 IRDataElement::GetName

HRESULT GetName(
    [out, string] LPOLESTR* ppwszName
);

Parameter

ppwszName

[out] Points to a location that upon successful completion points to a zero-terminated string containing the name of the data element. Upon failure, *ppwszName is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
Remarks

This method allocates memory for the string returned in the ppwszName parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.6.6 IRDataElement::SetAt

Sets a value within the data element.

HRESULT SetAt(
    [in] long cDim,
    [in, size_is(cDim)] long rgDim[],
    [in] RSOLVER_DATA data
);

Parameters

cDim

[in] Number of dimensions in the data element.

rgDim[]

[in] An array containing the index of each dimension that identifies which value to set within the data element. Element \( i \) in the array is the index for dimension number \( i \). This array has a size equal to the cDim parameter.

data

[in] An RSOLVER_DATA structure containing the new value of the item to set within the data element.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The item was successfully set.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>One of the indices in the rgDim array was invalid.</td>
</tr>
<tr>
<td>RSOLVE_E_READONLY</td>
<td>The data element is read only, and could not be modified.</td>
</tr>
<tr>
<td>RSOLVE_E_DATAELEMENTLOCKED</td>
<td>The data element was locked and could not be modified.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>
The operation can also return any of the error codes from the COM function VariantChangeType.

Remarks

The valid indices are specified by the valid indices for the dimension associated with the data element vector. See IRDimension for more details.

1.7 IRDataElement1

The interface IRDataElement1 provides optimized access to a data element that contains a vector value; that is, a value with a single dimension.

When to implement. Implementers of data elements will implement and expose this interface when the data element contains a vector value, if they wish to provide optimized access to the vector. In particular, solver outputs that are vectors, such as decision variables, are ideal candidates for implementation of IRDataElement1. This interface is in addition to IRDataElement, which must be implemented on all data elements.

When to use. Use this interface to optimize access to a data element if it has a vector value.

Methods in vtable order

IUnknown methods: Description
QueryInterface Returns pointers to supported interfaces.
AddRef Increments reference count.
Release Decrements reference count.
IRDataElement1 methods:
GetAt Returns an element in the vector.
SetAt Sets an element in the vector.

1.7.1 IRDataElement1::GetAt

Returns a value in the data element vector.

HRESULT GetAt( 
    [in] long Index, 
    [out, retval] RSOLVER_DATA* pData 
);

Parameters

Index
The index to retrieve the vector value for.

pData

Pointer to a caller-allocated RSOLVER_DATA structure that upon successful completion contains the value of the data element at the specified index.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The item was successfully retrieved.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The requested index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output pointer was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The valid indices are specified by the valid indices for the dimension associated with the data element vector. See IRDimension for more details.

1.7.2 IRDataElement1::SetAt

Sets a value in the data element vector.

HRESULT IRDataElement1::SetAt(
    [in] long Index,
    [in] RSOLVER_DATA data
);

Parameters

Index

[in] The index to set the vector value for.

data

[in] An RSOLVER_DATA structure that contains the new value to set the item in the vector to.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
</tbody>
</table>
IRDataElement2

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The requested index was invalid.</td>
</tr>
<tr>
<td>RSOLVE_E_READONLY</td>
<td>The data element is read-only, and could not be modified.</td>
</tr>
<tr>
<td>RSOLVE_E_DATAELEMENTLOCKED</td>
<td>The data element was locked, and could not be modified at this time.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

The method may also return any of the error codes from the COM function `VariantChangeType`.

Remarks

The valid indices are specified by the valid indices for the dimension associated with the data element vector. See `IRDimension` for more details.

1.8 IRDataElement2

The interface `IRDataElement2` provides optimized access to a data element that contains a matrix value; that is, a value with a two dimensions.

When to implement. Implementers of data elements will implement and expose this interface when the data element contains a matrix value, if they wish to provide optimized access to the matrix. This interface is in addition to `IRDataElement`, which must be implemented on all data elements.

When to use. Use this interface to optimize access to a data element if it is a (two-dimensional) matrix.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

IRDataElement2 methods:

| GetAt                           | Returns an element in the matrix.                                           |
| SetAt                           | Sets an element in the matrix.                                              |
| GetColumn                       | Returns a column in the matrix.                                             |
| GetRow                          | Returns a row in the matrix.                                                |

1.8.1 IRDataElement2::GetAt

Returns an item in the data element matrix.

```cpp
HRESULT GetAt(  
```
[in] long Row,
[in] long Column,
[out, retval] RSOLVER_DATA* pData
);

Parameters

Row

[in] Index of the row to retrieve the value for. Valid indices are defined by the dimension with index zero attached to this data element.

Column

[in] Index of the column to retrieve the value for. Valid indices are defined by the dimension with index one attached to this data element.

pData

[out] Pointer to a caller-allocated RSOLVER_DATA structure that upon successful completion contains the value at the specified row and column in the matrix.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSOLVE_E_INVALIDROW</td>
<td>The row index was invalid.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDCOLUMN</td>
<td>The column index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.8.2 IRDataElement2::GetColumn

Returns a column in the data element matrix as a separate data element object.

HRESULT GetColumn(
    [in] long Column,
    [in] DWORD dwCopyFlags
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppUnk
);
IRDataElement2

Parameters

Column

[in] Index of the column to retrieve as a separate data element. Valid indices are determined by the dimension with index one attached to the data element matrix.

dwCopyFlags

[in] Flags specifying whether to make a copy of the column or return the embedded column within the data element. These flags can be selected from the RSOLVE_DATA_ACCESS enumeration.

riid

[in] Identifier of the interface to acquire on the column data element.

ppUnk

[out] Pointer to location that upon successful completion contains the requested interface pointer.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSOLVE_E_NOTSUPPORTED</td>
<td>The data element matrix does not support sub-element access.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDCOLUMN</td>
<td>The column index was invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The data element does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

If the client passes the RDA_COPY flag in dwCopyFlags, this operation creates a new data element that contains a copy of the column (although the implementation can be optimized using copy-on-write semantics). If dwCopyFlags contains RDA_CONTAINED, then the returned data element is a reference to the column within the matrix, and changes to the new data element affect the matrix data element.

The function usually delegates to IRDataElementAccess::GetSubElementAt, if available.
1.8.3 IRDataElement2::GetRow

Returns a row in the data element matrix as a separate data element object.

```
HRESULT GetRow(
    [in] long Row,
    [in] DWORD dwCopyFlags
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppUnk
);
```

**Parameters**

**Row**

[in] Index of the row to retrieve as a separate data element. Valid indices are determined by the dimension with index zero attached to the data element matrix.

**dwCopyFlags**

[in] Flags specifying whether to make a copy of the row or return the embedded row within the data element. These flags can be selected from the RSOLVE_DATA_ACCESS enumeration.

**riid**

[in] Identifier of the interface to acquire on the row data element.

**ppUnk**

[out] Pointer to location that upon successful completion contains the requested interface pointer.

**Return values**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_NOTSUPPORTED</td>
<td>The data element matrix does not support sub-element access.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDROW</td>
<td>The row index was invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The data element does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
Remarks

If the client passes the RDA_COPY flag in dwCopyFlags, this operation creates a new data element that contains a copy of the row (although the implementation can be optimized using copy-on-write semantics). If dwCopyFlags contains RDA_CONTAINED, then the returned data element is a reference to the row within the matrix, and changes to the new data element affect the matrix data element.

The function usually delegates to IRDataElementAccess::GetSubElementAt, if available.

1.8.4 IRDataElement2::SetAt

Sets an item in the data element matrix.

HRESULT SetAt(
    [in] long Row,
    [in] long Column,
    [in] RSOLVER_DATA data
);

Parameters

Row

[in] Index of the row to set the value for. Valid indices are defined by the dimension with index zero attached to this data element.

Column

[in] Index of the column to set the value for. Valid indices are defined by the dimension with index one attached to this data element.

data

[in] An RSOLVER_DATA structure that contains the new value to set in the matrix.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDROW</td>
<td>The row index was invalid.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDCOLUMN</td>
<td>The column index was invalid.</td>
</tr>
<tr>
<td>RSOLVE_E_READONLY</td>
<td>The data element is read-only, and could not be modified.</td>
</tr>
<tr>
<td>RSOLVE_E_DATAELEMENTLOCKED</td>
<td>The data element was locked, and could not be modified.</td>
</tr>
</tbody>
</table>
The interface IRDataElementAccess provides sub-element access to a data element. Sub-element access is the process of returning as a separate data element a subset of the items in the data element, selected by fixing some dimension values and leaving the others free. See section 3.4.2.3.c of Ruark [1] for more details.

**When to implement.** Implement this optional interface on a data element in order to provide sub-element access, when necessary. This interface is never required for implementations that manage only scalar or vector data elements.

**When to use.** Use this interface to extract a sub-element from a data element, as a new data element. In particular, this interface provides for row or column access, or per-item queries of values.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRDataElementAccess method:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSubElementAt</td>
<td>Returns an embedded sub-matrix.</td>
</tr>
</tbody>
</table>

### IRDataElementAccess::GetSubElementAt

Returns a new data element containing the specified sub-element.

```c
HRESULT GetSubElementAt(
    [in] long cDim,
    [in, size_is(cDim)] long rgDim[],
    [in, size_is(cDim)] long rgIndex[],
    [in] DWORD dwCopyFlags
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppUnk
);
```

**Parameters**

- `cDim`
[in] Specifies the number of dimensions whose values will be fixed in the sub-element. This must be less than or equal to the number of dimensions in the original data element.

rgDim[]

[in] An array of cDim elements specifying, by index, which dimensions are being fixed. For instance, if cDim=3 and rgDim[] = {0, 2, 3} then the first, third, and fourth dimensions will be fixed by the values in rgIndex while the second dimension (and any others) will be unrestricted.

rgIndex[]

[in] An array of cDim elements specifying the values at which to fix the dimensions specified by rgDim[]. If cDim=3, rgDim[] = {0, 2, 3}, and rgIndex = {15, 2, -4}, then the new sub-element data element will have three fewer dimensions than the original data element, and each value will be calculated from the original by setting the first dimension to 15, the third to 2, and the fourth to -4.

dwCopyFlags

[in] Flags specifying whether to make a copy of the sub-element or return the embedded sub-element within the data element. These flags can be selected from the RSOLVE_DATA_ACCESS enumeration.

riid

[in] Identifier of the interface to acquire on the new data element.

ppUnk

[out] Pointer to location that upon successful completion contains the requested interface pointer.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>One of the column indices was invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The data element does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

If the client passes the RDA_COPY flag in dwCopyFlags, this operation creates a new data element that contains a copy of the sub-element (although the implementation can be optimized using copy-on-write semantics), and changes to the new data element do not affect the original data element. If
IRDataElementClone

If \( \text{dwCopyFlags} \) contains RDA_CONTAINED, then the returned data element is a reference to the sub-element within the matrix, and changes to the new data element affect the original data element.

1.10 IRDataElementClone

Interface IRDataElementClone enables a client to make a copy of a data element, creating a new data element with its own copy of the values from the original data element.

**When to implement.** Implement this interface on a data element object to provide native cloning capabilities. If this interface is not implemented, a client will have to manually copy the data element value by value to clone it.

**When to use.** Use this interface to make a clone of a data element using the data element’s implementation, which is usually optimized for the internal structure of the data element.

**Methods in vtable order**

**IUnknown methods:**
- **QueryInterface:** Returns pointers to supported interfaces.
- **AddRef:** Increments reference count.
- **Release:** Decrements reference count.

**IRDataElementClone method:**
- **Clone:** Clones the data element.

1.10.1 IRDataElementClone::Clone

Clones the data element and returns the new clone.

```cpp
HRESULT Clone( 
    [in] REFIID riid, 
    [out, iid_is{riid}] LPVOID* ppUnk 
);
```

**Parameters**

riid

[in] Identifier of the interface to return of the newly created clone.

ppUnk

[out] Pointer to location that upon successful completion contains the requested interface pointer of the cloned data element.

**Return values**

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
</table>
IRDataElementCreator

1.11 IRDataElementCreator

The interface IRDataElementCreator enables a client to create a new data element. This is not the only way that data elements may be created. Solvers that output data elements may create those data element using internal techniques specific to their implementation language (by using the operator new in C++, for instance). Solvers may also use IRDataElementCreator internally to create their output data elements.

When to implement. Data element factories implement this interface to provide for the creation of data elements.

When to use. Use this interface to create a data element. Generally, objects that use their own implementation of data elements may create those data element using internal techniques. When they use data elements implemented by other modules, they should use IRDataElementCreator or another suitable interface.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

IRDataElementCreator method:

| Create | Creates a data element. |

1.11.1 IRDataElementCreator::Create

Creates a new data element with a specified name, number of dimensions and associated dimension objects, and data type.

HRESULT Create(
    [in, string] LPOLESTR pwszName,
    [in] long cDim,
    [in, size_is(cDim)] LPRDIMENSION* rgpDim,
    [in] RSOLVER_DATATYPE type,
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppUnk
);
IRDataElementCreator

Parameters

pwszName

[in] Pointer to a zero-terminated string containing the name of the new data element.

cDim

[in] The number of dimensions the new data element will have.

rgpDim

[in] An array of IRDimension pointers that identify, in order, the dimension objects that define the
range of the new data element.

type

[in] An RSOLVER_DATATYPE enumeration value (VARTYPE) indicating the type of data to be stored
in the data element.

riid

[in] Identifier of the interface to return from the new data element. Typically, this is
IID_IRDataElement.

ppUnk

[out] Pointer to a location that upon successful creation of the data element contains the requested
interface pointer of the new data element.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_WRONGNUMBEROF DIMENSIONS</td>
<td>The data element implementation does not support the requested number of dimensions.</td>
</tr>
<tr>
<td>RSOLVE_E_WRONGDATATYPE</td>
<td>The data element implementation does not support the requested data type.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>One of the input parameters was invalid.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
IcadelementitemProperties

Remarks

Some implementations of data elements may support a fixed or limited number of dimensions or
data types. These implementations may be optimized for their particular data type. For instance, a
sparse matrix implementation of doubles may only allow two dimensions and doubles to be stored as
their data type. Hence, the creator has the option of failing the creation with the errors
RSOLVE_E_WRONGNUMBEROFDIMENSIONS or RSOLVE_E_WRONGDATATYPE in these cases.

1.12 IcadelementitemProperties

Interface IcadelementitemProperties enables a client to access item properties associated with each
value in a data element. Item properties are name-value pairs on values in data elements. Each value
in a data element supports a set of name-value pairs.

When to implement. Implement this interface on a data element object if it supports item
properties.

When to use. Use this interface to access the item properties of values in a data element.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

IRDataelementitemProperties methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetItemPropertyCount</td>
<td>Returns number of properties.</td>
</tr>
<tr>
<td>GetItemPropertyName</td>
<td>Returns name of a property.</td>
</tr>
<tr>
<td>MapItemPropertyNameToId</td>
<td>Maps a property name to a property ID.</td>
</tr>
<tr>
<td>GetItemProperty</td>
<td>Returns a property value.</td>
</tr>
</tbody>
</table>

1.12.1 IRDataelementitemProperties::GetItemProperty

Returns an item property for a specific value within the data element.

HRESULT GetItemProperty(
    [in] UINT uPropID,
    [in] long cDim,
    [in, size_is(cDim)] long rgDim[],
    [out] VARIANT* pProperty
);

Parameters

uPropID

[in] Index of the item property to return. This value must be less than the value returned from
GetItemPropertyCount.
cDim

[in] Number of dimensions in the data element.

rGDim

[in] An array with cDim elements identifying the particular value within the data element to retrieve the item property for. See IRDataElement::GetAt for more information.

pProperty

[out] Pointer to a caller-allocated VARIANT that upon successful completion contains the requested item property.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_WRONGNUMBEROF-</td>
<td>The dimension count was invalid for this data</td>
</tr>
<tr>
<td>DIMENSIONS</td>
<td>element.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>One of the dimension indices was invalid.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The item property index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.12.2 IRDataElementItemProperties::GetItemPropertyCount

Returns the number of item properties contained in the data element. Each value in the data element exposes the same item properties.

```cpp
HRESULT GetItemPropertyCount(
    [out] UINT* puCount
);
```

Parameter

puCount

[out] Pointer to a UINT that upon successful completion contains the number of item properties contained in the data element. Upon failure, *puCount is set to zero.
IRDataElementItemProperties

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.12.3 IRDataElementItemProperties::GetItemPropertyName

Returns the name of an item property.

```
HRESULT GetItemPropertyName(
    [in] UINT uPropID,
    [out, string] LPOLESTR* ppwszName
);
```

Parameters

- **uPropID**
  - [in] Index of the item property for which to return the name. This value must be less than the value returned from GetItemPropertyCount.

- **ppwszName**
  - [out] Pointer to location that upon successful completion points to a zero-terminated string containing the name of the item property. Upon failure, *ppwszName is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The item property index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This method allocates memory for the string returned in the ppwszName parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.
1.12.4 IRDataElementItemProperties::MapItemPropertyNameTold

Maps an item property name into the item property's index.

```c
HRESULT MapItemPropertyNameTold(
    [in, string] LPOLESTR pwszName,
    [out] UINT* pPropID
);
```

**Parameters**

- **pwszName**
  - [in] Pointer to a zero-terminated string containing the name of the item property.
- **pPropID**
  - [out] Pointer to a UINT that upon successful completion contains the index of the named item property. Upon failure, *pPropID is set to zero.

**Return values**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The requested item property name was not a valid item property name for this data element.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.13 IRDataElementItemPropertiesChange

Interface IRDataElementItemPropertiesChange enables a client to change the per-value item properties of a data element.

**When to implement.** Implement this interface on a data element that supports item properties that can be modified by the client.

**When to use.** Use this interface to change item properties on a data element. Normally, this happens during the data analysis and pre-processing phases, as item properties are not intended to be active, modified values during the processing of a solution network.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
</table>
IRDataElementItemPropertiesChange

QueryInterface returns pointers to supported interfaces.
AddRef increments reference count.
Release decrements reference count.
IRDataElementItemPropertiesChange method:
SetItemProperty sets a property item value.

1.13.1 IRDataElementItemPropertiesChange::SetItemProperty

Changes the value of an item property for a specific value within the data element.

HRESULT SetItemProperty(
    [in] UINT uProplD,
    [in] long cDim,
    [in, size_is(cDim)] long rdDim[],
    [in] VARIANT Property
);

Parameters

uProplD
[in] Index of the item property to change. This value must be less than the value returned from GetItemPropertyCount.

cDim
[in] Number of dimensions in the data element.

rgDim
[in] An array with cDim elements identifying the particular value within the data element to change the item property for. See IRDataElement::GetAt for more information.

Property
[in] The new value of the item property.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_WRONGNUMBEROF-DIMENSIONS</td>
<td>The dimension count was invalid for this data element.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>One of the dimension indices was invalid.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The item property index was invalid.</td>
</tr>
</tbody>
</table>
The interface `IRDataElementScalar` provides optimized access to a data element that contains a scalar, or dimensionless, value.

**When to implement.** Implementers of data elements will implement and expose this interface when the data element contains a scalar value, if they wish to provide optimized access to the scalar value. In particular, solver outputs that are scalars, such as optimal values, are ideal candidates for implementation of `IRDataElementScalar`. This interface is in addition to `IRDataElement`, which must be implemented on all data elements.

**When to use.** Use this interface to optimize access to a data element if it has a scalar value.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

**IRDataElementScalar methods:**

| GetScalar          | Returns the scalar value. |
| SetScalar          | Sets the scalar value. |

### 1.14.1 IRDataElementScalar::GetScalar

Returns the value of the scalar data element.

```c
HRESULT GetScalar(
    [out, retval] RSOLVER_DATA* pData
);
```

**Parameter**

- `pData`

  `[out]` Pointer to a caller-allocated RSOLVER_DATA structure that upon successful completion contains the scalar value of the data element.

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>
IRDataSource

<table>
<thead>
<tr>
<th>E_POINTER</th>
<th>The parameter was an invalid pointer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.14.2 IRDataElementScalar::SetScalar

Sets the scalar value of the data element.

```c++
HRESULT SetScalar(
    [in] RSOLVER_DATA data
);
```

Parameter

data

[in] An RSOLVER_DATA structure containing the new value to assign to the data element.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSOLVE_E_READONLY</td>
<td>The data element is read-only, and could not be modified.</td>
</tr>
<tr>
<td>RSOLVE_E_DATAELEMENTLOCKED</td>
<td>The data element was locked, and could not be modified at this time.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

This method may also return any of the error codes from the COM function VariantChangeType.

1.15 IRDataSource

Interface IRDataSource is the primary interface of data sources within the framework. A data source is the owner or provider of multiple data elements. Whereas data elements manage the actual, static values of data elements, data sources manage the dynamic life cycles of the data elements they manage.

**When to implement.** Implement this interface on any data source object. In particular, solver outputs are data elements, and typically a solver will implement this interface to consolidate the outputs of the solver under a single data source.
When to use. Use this interface to acquire data elements managed by a data source, to establish and destroy data advisory connections, and to lock and unlock the data elements managed by a data source.

Methods in vtable order

IUnknown methods:
- QueryInterface
- AddRef
- Release

IRDataSource methods:
- Advise
- Unadvise
- LockData
- UnlockData
- DataLocked
- GetDataElementCount
- GetDataElementName
- GetDataElementAccessor
- GetDataElementModifier
- CanMaintainSolverData
- LockRequired

1.15.1 IRDataSource::Advise

Called by an object supporting a data advise sink to create a connection between a data source and the advise sink. This enables the advise sink to be notified of changes in the data of the data elements managed by the data source.

HRESULT Advise(
    [in] LPRDATAADVISE pAdvise,
    [out] DWORD* pdwCookie
);

Parameters

pAdvise

[in] Pointer to the IRDataAdvise sink that will receive data advise notifications.

pdwCookie

[out] Pointer to a DWORD cookie that identifies this connection. Use this cookie later to delete the advisory connection (by passing it to IRDataSource::Unadvise). If this value is zero, the connection was not established.
IRDataSource

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The advisory connection was established.</td>
</tr>
<tr>
<td>OLE_E_ADVISENOTSUPPORTED</td>
<td>The data source does not support data advise notifications.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The advise sink pointer was invalid.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Note to implementers

To simplify the implementation of Advise and the other notification methods in IRDataSource, the framework provides an advise holder object that manages the registration and sending of notifications. To get a pointer to this object, create the DataAdviseHolder object (CLSID CLSID_RDataAdviseHolder) on the first invocation of Advise. This supplies a pointer to the object's IRDataAdviseHolder interface. Then, delegate the call to the IRDataAdviseHolder::Advise method in the data advise holder, which creates, and subsequently manages, the requested connection.

1.15.2 IRDataSource::CanMaintainSolverData

Returns whether the data source can maintain a solver's inputs during the execution of a solver. This is usually possible, unless the underlying data set is volatile.

```c
HRESULT CanMaintainSolverData(
    [out] DWORD* pfCan
);
```

Parameter

pfCan

[out] Pointer to a DWORD that upon successful completion is non-zero if the data source can maintain the data for the solver or zero if it cannot. Upon failure, *pfCan is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
1.15.3 IRDataSource::DataLocked

Returns whether the data source is currently locked or not.

```cpp
HRESULT DataLocked(
    [out] DWORD* pfLocked
);
```

Parameter

`pfLocked`

[out] Pointer to a DWORD that upon successful completion is zero if the data source is not locked, and non-zero if the data source is locked. This value is set to zero if the operation fails.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The data source maintains a (non-negative) lock count, which is incremented by calls to `IRDataSource::LockData` and decremented by calls to `IRDataSource::UnlockData`. The data source is locked if the lock count is non-zero.

1.15.4 IRDataSource::GetDataElementAccessor

Returns the accessor interface of one of the data elements managed by the data source. An accessor interface allows the client to query values within the data element, but does not permit changes to the data element.

```cpp
HRESULT GetDataElementAccessor(
    [in] UINT ulndex,
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);
```

Parameters

`ulndex`

[in] Index identifying which of the data elements managed by the data source will be returned. This number must be less than the value returned by `GetDataElementCount`. 
IRDataSource

riid

[in] Identifier of the accessor interface to return of the managed data element.

ppVoid

[out] Pointer to a location that upon successful completion contains a pointer to the requested interface. Upon failure, *ppVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The index was invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The data element accessor does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.15.5 IRDataSource::GetDataElementCount

Returns the total number of data elements managed by this data source.

```cpp
HRESULT GetDataElementCount(
    [out] UINT* puCount
);
```

Parameter

puCount

[out] Pointer to a UINT that upon successful completion contains the number of data elements managed by the data source. On failure *puCount is set to 0.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
1.15.6 IRDataSource::GetDataElementModifier

Returns a modifier interface of one of the data elements managed by the data source. The modifier interface enables a client to change values within the data element.

HRESULT GetDataElementModifier(
    [in] UINT ulIndex,
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);

Parameters

ulIndex

[in] Index identifying which of the data elements managed by the data source will be returned. This number must be less than the value returned by GetDataElementCount.

riid

[in] Identifier of the modifier interface to return of the managed data element.

ppVoid

[out] Pointer to a location that upon successful completion contains a pointer to the requested interface. Upon failure, *ppVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The index was invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The data element modifier does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.15.7 IRDataSource::GetDataElementName

Returns the name of a data element managed by the data source.

HRESULT GetDataElementName(
    [in] UINT ulIndex,
...
IRDataSource

[out, retval] BSTR* pbstr

Parameters

ulndex

[in] The index of the data element to retrieve the name for. This value must be between zero and one less than the value returned from IRDataSource::GetDataElementCount.

pbstr

[out] Points to a location that upon successful completion points to a BSTR containing the name of the data element. Upon failure, *pbstr is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This method allocates memory for the string returned in the pbstr parameter using the SysAllocString function. The caller is responsible for calling SysFreeString to free the string.

This function is similar to IRDataElement::GetName.

1.15.8 IRDataSource::LockData

Locks all of the data elements managed by the data source. The data elements will be prevented from changing their values.

HRESULT LockData();

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
Remarks

This is typically implemented using a lock count, so that LockData increments the count while UnlockData decrements the count. The data source is locked when the count is not zero.

1.15.9 IRDataSource::LockRequired

Returns whether the data source must be locked in order for it to maintain the data for a solver during solver execution. This value is valid only if IRDataSource::CanMaintainSolverData returns a non-zero value. If a lock is required, the client, mapping, inbound solver site, or solver must call IRDataSource::Lock before using the data source and IRDataSource::Unlock after using it.

HRESULT LockRequired(
    [out] DWORD* pfRequired
);

Parameter

pfRequired

[out] Pointer to a DWORD which upon successful completion contains a non-zero value if a lock is required, or zero if a lock is not required. Upon failure, *pfRequired is set to zero.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.15.10 IRDataSource::Unadvise

Destroys an advisory notification connection.

HRESULT Unadvise(
    [in] DWORD dwCookie
);

Parameter

dwCookie

[in] A cookie that identifies the connection to destroy. This cookie must have been previously obtained from a matching call to IRDataSource::Advise.
IRDescriptionPropEnumeration

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The connection was successfully terminated.</td>
</tr>
<tr>
<td>OLE_E_NOCONNECTION</td>
<td>The specified dwCookie is not a valid connection.</td>
</tr>
<tr>
<td>OLE_E_ADVISENOTSUPPORTED</td>
<td>This IRDataSource implementation does not support notification.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This method destroys a notification created with a call to the IRDataSource::Advise method.

If the advisory connection being deleted was initially set up by delegating the IRDataSource::Advise call to IRDataAdviseHolder::Advise, this call must be delegated to IRDataAdviseHolder::Unadvise to delete it.

1.15.11 IRDataSource::UnlockData

Unlocks all of the data elements managed by the data source, which had previously been locked through a call to LockData.

HRESULT UnlockData();

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_UNEXPECTED</td>
<td>The data source was not locked.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This is typically implemented using a lock count, so that LockData increments the count while UnlockData decrements the count. The data source is locked when the count is not zero.

1.16 IRDescriptionPropEnumeration

The interface IRDescriptionPropEnumeration enables a client to enumerate the description properties that are registered with the core services, or to lookup a particular description property in the list of registered description properties. This interface is implemented by the core services object DescriptionPropManager, which has CLSID CLSID_RDescriptionPropManager and can be created through a normal call to the COM function CoCreateInstance.
When to implement. It is not necessary to implement this interface, except in the case of implementing a new description property manager. This interface is implemented by the core services object DescriptionPropManager.

When to use. Use this interface to enumerate the registered description properties, or to lookup the name of a particular description property.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRDescriptionPropEnumeration methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnumRDESCPROPIDINFO</td>
</tr>
<tr>
<td>LookupDescription</td>
</tr>
</tbody>
</table>

1.16.1 IRDescriptionPropEnumeration::EnumRDESCPROPIDINFO

Returns an enumerator object for the description property information structures RDESCPROPIDINFO for a given locale.

```c
HRESULT EnumRDESCPROPIDINFO(
    [in] LCID Icid,
    [out] IEnumRDESCPROPIDINFO** ppenumDesclnfo
);
```

Parameters

<table>
<thead>
<tr>
<th>Icid</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] Locale identifier to enumerate description properties for.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ppenumDesclnfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out] Pointer to a location that on successful completion contains a pointer to the enumeration interface IEnumRDESCPROPIDINFO, which is a standard COM enumerator interface of RDESCPROPIDINFO structures.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
IRDescriptionPropEnumeration

1.16.2 IRDescriptionPropEnumeration::LookupDescription

Looks up the name of a given RDESCPROPID description property identifier within the list of registered description properties, for a given locale.

```cpp
HRESULT LookupDescription(
    [in] REFGUID rguid,
    [in] LCID lcid,
    [out, string] LPOLESTR* ppwszName
);```

Parameters

rguid

[in] The RDESCPROPID to look up the name for.

lcid

[in] The locale identifier for which to return the name.

ppwszName

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the name of the description property. Upon failure, *ppwszName is set to zero.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>REGDB_E_CLASSNOTREG</td>
<td>The description property ID has not been registered with the core services.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This method allocates memory for the string returned in the ppwszName parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.
1.17 **IRDescriptionPropRegistration**

Interface IRDescriptionPropRegistration enables a solver, SolverInfo object, or client to register and unregister description property identifiers (RDESCPROPIDs) with the core services system.

RDESCPROPIDs are GUIDs that identify a particular description property, as exposed by SolverInfo objects through the IRSolverDescriptionProperties interface. RDESCPROPIDs for the methods defined in IRSolverDescription are registered with the system during installation of the core services. Solvers that wish to define description properties beyond the standard ones should register their property identifiers with the core services through IRDescriptionPropRegistration.

**When to implement.** It is not necessary to implement this interface. This interface is provided by the core services DescriptionPropManager object, which can be created by calling the COM function CoCreateInstance with a CLSID of CLSID_RDescriptionPropManager.

**When to use.** Use this interface during the installation or removal of a solver that defines its own solver description property identifiers to register or unregister them with the core services. This interface is provided by the core services DescriptionPropManager object, which can be created by calling the COM function CoCreateInstance with a CLSID of CLSID_RDescriptionPropManager.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
<tr>
<td>IRDescriptionPropRegistration</td>
<td></td>
</tr>
<tr>
<td>RegisterRDESCPROPID</td>
<td>Registers a description property with the system.</td>
</tr>
<tr>
<td>UnregisterRDESCPROPID</td>
<td>Unregisters a description property with the system.</td>
</tr>
</tbody>
</table>

### 1.17.1 IRDescriptionPropRegistration::RegisterRDESCPROPID

 Registers a description property identifier with the core services.

```c
HRESULT RegisterRDESCPROPID(
    [in] REFGUID rguid,
    [in] LCID Icid,
    [in, string] LPOLESTR pwszName
);
```

**Parameters**

- `rguid`
- `Icid`
  - [in] The locale identifier under which to register the name of this description property.
IRDescriptionPropRegistration

pwszName

[in] Pointer to a zero-terminated string containing the name of the description property, for the specified locale.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>One of the input parameters was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed for an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

Description property names are specialized for locales, so the client should register a description property name for each locale that will be used.

1.17.2 IRDescriptionPropRegistration::UnregisterRDESCPROPID

Unregisters a description property from the core services.

HRESULT UnregisterRDESCPROPID(
    [in] REFGUID rguid,
    [in] LCID Icid
);

Parameters

rguid

[in] The description property identifier to unregister.

Icid

[in] The locale identifier to unregister.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>One of the input parameters was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed for an unspecified reason.</td>
</tr>
</tbody>
</table>
1.18 IRDimension

IRDimension is the primary dimension interface. A dimension is an association object that ties together a data element and a set. A data element may contain many dimensions, each one of which is associated with a single set. A single set may be associated with many dimensions. Dimensions are essentially place-holders for indices into a data element matrix.

When to implement. Implement this interface on any dimension object.

When to use. Use this interface to access the underlying index bounds and set information on a dimension within a data element.

Methods in vtable order

IUnknown methods:  
- QueryInterface  
- AddRef  
- Release  

IRDimension methods:  
- GetName  
- GetBounds  
- GetSet

1.18.1 IRDimension::GetBounds

Returns the lower and upper bounds of indices used by items in the set associated with the dimension.

HRESULT GetBounds(  
    [out] long* plLower,  
    [out] long* plUpper  
);

Parameters

plLower

[out] Pointer to a long which upon successful completion contains the lowest index used by items in the dimension. Upon failure, *plLower is set to zero.

plUpper

[out] Pointer to a long which upon successful completion contains the highest index used by items in the dimension. Upon failure, *plUpper is set to zero.

Return values

| S_OK | The operation completed successfully. |
IRDimension

<table>
<thead>
<tr>
<th>E_POINTER</th>
<th>The output parameter was an invalid pointer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This function usually delegates to IRSet::GetSetBounds if there is a set associated with the dimension.

1.18.2 IRDimension::GetName

Returns the name of the dimension.

```
HRESULT GetName(
    [out, string] LPOLESTR* ppwszName
);
```

Parameter

ppwszName

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the name of the dimension. Upon failure, *ppwszName is set to zero.

Return values.

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed for an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This method allocates memory for the string returned in the ppwszName parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.18.3 IRDimension::GetSet

Returns the set that defines the valid domain indices of the dimension, if there is an attached set.

```
HRESULT GetSet(
    [in] REFIID riid,
```
IRDimensionClone

[out, iid_is(riid)] LPVOID* ppUnk

Parameters

riid

[in] Identifier of the interface to return on the attached set.

ppUnk

[out] Pointer to a location that upon successful completion points to the requested interface of the attached set. Upon failure, *ppUnk is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOINTERFACE</td>
<td>The set does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed for an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

If there is no attached set (i.e., the dimension was created from a range), this function should set *ppUnk to zero.

1.19 IRDimensionClone

Interface IRDimensionClone enables a client to make a copy of a dimension, creating a new dimension with its own copy of the values from the original dimension.

When to implement. Implement this interface on a dimension object to provide native cloning capabilities. If this interface is not implemented, a client will have to manually copy the dimension to clone it.

When to use. Use this interface to make a clone of a dimension using the dimension's implementation, which is usually optimized for the internal structure of the dimension. In particular, use this interface during the implementation of IRDataElementClone on data elements.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
</tbody>
</table>
IRDimensionCreator

AddRef
Release

IRDimensionClone method:
Clone

Increments reference count.
Decrements reference count.
Clones the dimension object.

1.19.1 IRDimensionClone::Clone

Clones the dimension and returns the new clone.

HRESULT Clone(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppUnk
);

Parameters

riid

[in] Identifier of the interface to return of the newly created clone.

ppUnk

[out] Pointer to location that upon successful completion contains the requested interface pointer of the cloned dimension. Upon failure, *ppUnk is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOINTERFACE</td>
<td>The clone does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.20 IRDimensionCreator

The interface IRDimensionCreator enables a client to create a new dimension. This is not the only way that dimensions may be created. Solvers that output data elements may create the dimensions for those data elements using internal techniques specific to their implementation language (by using operator new in C++, for instance). Solvers may also use IRDimensionCreator internally to create those dimensions.
A dimension is usually created in this manner as a preliminary step to creating a data element that uses the dimension.

**When to implement.** Dimension factories implement this interface to provide for the creation of dimensions.

**When to use.** Use this interface to create a dimension. Generally, objects that use their own implementation of dimensions may create those dimensions using internal techniques. When they use dimensions implemented by other modules, they should use IRDimensionCreator or another suitable interface.

**Methods in vtable order**

**IUnknown methods:**

- QueryInterface: Returns pointers to supported interfaces.
- AddRef: Increments reference count.
- Release: Decrements reference count.

**IRDimensionCreator methods:**

- CreateFromRange: Creates a dimension from a range of indices.
- CreateFromSet: Creates a dimension from a set.

### 1.20.1 IRDimensionCreator::CreateFromRange

Creates a new dimension object given a range of indices. The dimension will create and manage its own set object that contains the values in the range of indices.

```c
HRESULT CreateFromRange(
    [in, string] LPOLESTR pwszName,
    [in] long Lower,
    [in] long Upper,
    [in] REFIID riid,
    [out, iid_is{riid}] LPVOID* ppUnk
);
```

**Parameters**

- **pwszName**
  
  [in] Pointer to a zero-terminated string containing the name of the new dimension object.

- **Lower**

  [in] The lower bound of the range.

- **Upper**

  [in] The upper bound of the range.

- **riid**
**IRDimensionCreator**

[in] Identifier of the interface to return on the newly created dimension object.

ppUnk

[out] Pointer to location that upon successful completion contains the requested interface pointer of the newly created dimension object. Upon failure, *ppUnk is set to zero.

**Return values**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The name was invalid or the lower and upper bounds were invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The dimension object does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

The set that the dimension object uses has no intrinsic meaning; it is simply a placeholder. This method enables the creation of dimensions that have no associated meaningful set, and is useful for creating dimensions that simply express a range of values.

**1.20.2 IRDimensionCreator::CreateFromSet**

Creates a new dimension object from an existing set. The dimension will maintain a reference count on the set, and use it to determine the cardinality of the dimension.

```c
HRESULT CreateFromSet(
    [in, string] LPCTSTR pwszName,
    [in] LPUNKNOWN pUnk,
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppUnk
);
```

**Parameters**

pwszName

[in] Pointer to a zero-terminated string containing the name of the new dimension object.

pUnk
IRMappingAdvise

[in] The IUnknown pointer of the set on which the dimension will be based.

riid

[in] Identifier of the interface to return on the newly created dimension object.

ppUnk

[ out] Pointer to location that upon successful completion contains the requested interface pointer of
the newly created dimension object. Upon failure, *ppUnk is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The name or the set was invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The dimension object does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.21 IRMappingAdvise

Interface IRMappingAdvise is used by mappings to notify the inbound solver site that the data
managed by the mapping is available from its data source.

When to implement. Implement this interface on a sub-object of an inbound solver site, to receive
notifications from mappings.

When to use. Mappings use this interface to notify inbound solver sites that their data is available
for mapping. Clients could also use this interface to kick-start the solution network. Otherwise, there
is no need to use this interface.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
<tr>
<td>IRMappingAdvise method:</td>
<td></td>
</tr>
<tr>
<td>OnMappingDataAvailable</td>
<td>The mapping’s data is available.</td>
</tr>
</tbody>
</table>

53
1.21.1 IRMappingAdvise::OnMappingDataAvailable

Notifies an inbound solver site that the data managed by a mapping is available and ready to be mapped.

```vbnet
HRESULT OnMappingDataAvailable(
    [in] DWORD dwCookie
);
```

**Parameter**

dwCookie

[in] A cookie value that identifies the mapping for which data is available. This cookie is returned from the method IRSolverSiteInMappings::AddMapping.

**Return values**

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_INVALIDARG</td>
<td>The dwCookie was not a valid cookie for this inbound solver site.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.22 IRSet

IRSet is the primary interface for sets within the framework. It enables access to the values of a set, as well as the ability to lock and unlock the set.

**When to implement.** Implement this interface on set objects in the framework.

**When to use.** Use this interface to access the values of a set, or to lock or unlock a set.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRSet methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetName</td>
<td>Returns the name of the set.</td>
</tr>
<tr>
<td>GetCount</td>
<td>Returns the size of the set.</td>
</tr>
<tr>
<td>GetItem</td>
<td>Returns an item in the set.</td>
</tr>
<tr>
<td>IsMember</td>
<td>Returns whether an item is in the set.</td>
</tr>
<tr>
<td>MapToIndex</td>
<td>Maps an item to its index.</td>
</tr>
<tr>
<td>GetSetBounds</td>
<td>Returns the index bounds of the set.</td>
</tr>
</tbody>
</table>
IRSet

LockSet
UnLockSet
SetLocked

1.22.1 IRSet::GetCount

Returns the number of items in the set.

HRESULT GetCount(
    [out, retval] ULONG* puCount
);

Parameter

puCount

[out] Pointer to a ULONG that upon successful completion contains the number of items in the set. On error, *puCount is set to zero.

Return values

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed for an unspecified reason.</td>
</tr>
</tbody>
</table>

1.22.2 IRSet::GetItem

Returns an item, looked up by index, from the set.

HRESULT GetItem(
    [in] long Index,
    [out, retval] VARIANT* ptem
);

Parameters

Index

[in] Index of the item to return.

ptem

[out] Pointer to a caller-allocated VARIANT that upon successful completion contains the item.
IRSet

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.22.3 IRSet::GetName

Returns the name of the set.

HRESULT GetName(
    [out, string] IPOLESTR* ppwszName
);

Parameter

ppwszName

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the name of the set. Upon failure, *ppwszName is set to zero.

Return values.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This method allocates memory for the string returned in the ppwszName parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.22.4 IRSet::GetSetBounds

Returns the lower and upper bounds of indices used by items in the set.
HRESULT GetSetBounds(  
    [out] long* plLower,  
    [out] long* plUpper  
);  

Parameters  

plLower  

[out] Pointer to a long that upon successful completion contains the lowest index used by items in  
the set. Upon failure, *plLower is set to zero.  

plUpper  

[out] Pointer to a long that upon successful completion contains the highest index used by items in  
the set. Upon failure, *plUpper is set to zero.  

Return values  

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>One of the output parameters was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.22.5 IRSet::IsMember  

Returns whether an item is a member of the set.  

HRESULT IsMember(  
    [in] VARIANT Item,  
    [out, retval] VARIANT_BOOL* pbMember  
);  

Parameters  

Item  

[in] A VARIANT containing the item to check for membership.  

pbMember  

[out] Pointer to a VARIANT_BOOL that upon successful completion is VARIANT_TRUE if Item is a  
member of the set, and VARIANT_FALSE if Item is not. Upon failure, *pbMember is set to  
VARIANT_FALSE.
1.22.6 IRSet::LockSet

Locks the set, and prevents changes to the set.

HRESULT LockSet();

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_INVALIDARG</td>
<td>The Item parameter was a malformed VARIANT.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This is usually implemented using a lock count. LockSet increments the lock count, while UnlockSet decrements it. The set is locked when the count is non-zero.

1.22.7 IRSet::MapToIndex

Maps an item in the set to its index, if it is a member of the set.

HRESULT MapToIndex(
    [in] VARIANT Item,
    [out, retval] long* plIndex
);

Parameters

Item

[in] A VARIANT containing a member of the set.

plIndex

[out] Pointer to a long that upon successful completion contains the index of the Item in the set. Upon failure, *plIndex is set to zero.
Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The Item parameter is a malformed VARIANT or is not a member of the set.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.22.8 IRSet::SetLocked

Returns whether the set is currently locked or not.

```c
HRESULT SetLocked(
    [out] VARIANT_BOOL* pbLocked
);
```

Parameter

`pbLocked`

[out] Pointer to a VARIANT_BOOL that upon successful completion is VARIANT_FALSE if the set is not locked, and VARIANT_TRUE if the set is locked. This value is set to zero if the operation fails.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The set maintains a (non-negative) lock count, which is incremented by calls to IRSet::LockSet and decremented by calls to IRSet::UnlockSet. The set is locked if the lock count is non-zero.

1.22.9 IRSet::UnlockSet

Unlocks the set.

```c
HRESULT UnlockSet();
```
IRSetClone

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_UNEXPECTED</td>
<td>The set was not locked.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This is usually implemented using a lock count. LockSet increments the lock count, while UnlockSet decrements it. The set is locked when the count is non-zero.

1.23 IRSetClone

Interface IRSetClone enables a client to make a copy of a set, creating a new set with its own copy of the values from the original set.

When to implement. Implement this interface on a set object to provide native cloning capabilities. If this interface is not implemented, a client will have to manually copy the set to clone it.

When to use. Use this interface to make a clone of a set using the set’s implementation, which is usually optimized for the internal structure of the set. In particular, use this interface during the implementation of IRDimensionClone on dimensions.

Methods in vtable order

IUnknown methods:
- QueryInterface: Returns pointers to supported interfaces.
- AddRef: Increments reference count.
- Release: Decrements reference count.

IRSetClone method:
- Clone: Clones the set object.

1.23.1 IRSetClone::Clone

Clones the set and returns the new clone.

HRESULT Clone(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppUnk
);

Parameters

riid

[in] Identifier of the interface to return of the newly created clone.
IRSetCreator

_ppUnk

[out] Pointer to location that upon successful completion contains the requested interface pointer of
the cloned set. Upon failure, *ppUnk is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The clone does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.24 IRSetCreator

The interface IRSetCreator enables a client to create a new set. This is not the only way that sets may
be created. Solvers that output data elements may create the sets for those data elements using
internal techniques specific to their implementation language (by using operator new in C++, for
instance). Solvers may also use IRSetCreator internally to create those sets.

A set is usually created in this manner as a preliminary step to creating a dimension that uses the set,
or as an early step in the solution process.

When to implement. Set factories implement this interface to provide for the creation of sets.

When to use. Use this interface to create a set. Generally, objects that use their own implementation
of sets may create those sets using internal techniques. When they use sets implemented by other
modules, they should use IRSetCreator or another suitable interface.

Methods in vtable order

IUnknown methods:                          Description
   QueryInterface                          Returns pointers to supported interfaces.
   AddRef                                 Increments reference count.
   Release                                Decrements reference count.
IRSetCreator method:
   Create                               Creates a new set object.

1.24.1 IRSetCreator::Create

Creates a new, empty set with the specified name and underlying data type.

HRESULT Create(
IRSetModifier

[in, string] LPOLESTR pwszName,
[in] RSOLVER_DATATYPE type,
[in] REFIID riid,
[out, iid_is(riid)] LPVOID* ppUnk

Parameters

pwszName

[in] Pointer to a zero-terminated string containing the name of the new set.

type

[in] The type of the underlying data type to store in the set.

riid

[in] Identifier of the interface to return on the newly created set.

ppUnk

[out] Pointer to a location that upon successful completion contains a pointer to the requested interface. Upon failure, *ppUnk is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>One of the input parameters was invalid.</td>
</tr>
<tr>
<td>RSOLVE_E_WRONGDATATYPE</td>
<td>The implementation of the set does not support the requested data type.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.25 IRSetModifier

Interface IRSetModifier enables a client to change the values within a set.

When to implement. Implement this interface on a set object that supports changes to the set.

When to use. Use this interface to change the values within a set. Clients typically use this interface to modify exogenous set objects during the modeling or pre-processing phases of a solution network.
Methods in vtable order

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IUnknown</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>QueryInterface</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Decrements reference count.</td>
</tr>
<tr>
<td>Release</td>
<td></td>
</tr>
<tr>
<td>IRSetModifier</td>
<td>Adds an item to the set.</td>
</tr>
<tr>
<td>Insert</td>
<td>Sets an item in the set by index.</td>
</tr>
<tr>
<td>Set</td>
<td>Changes an item to another item in the set.</td>
</tr>
<tr>
<td>Change</td>
<td>Empties the set.</td>
</tr>
<tr>
<td>Clear</td>
<td>Removes an item from the set by value.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes an item from the set by index.</td>
</tr>
<tr>
<td>RemoveAt</td>
<td></td>
</tr>
</tbody>
</table>

1.25.1 IRSetModifier::Change

Changes an item in the set, without modifying the index.

```c
HRESULT Change(
    [in] VARIANT FromItem,
    [in] VARIANT Toltem
);
```

Parameters

FromItem

[in] Item to change within the set.

Toltem

[in] New value to assign to the old item.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>Either of the inputs was an malformed VARIANT, or the FromItem parameter is not a member of the set.</td>
</tr>
<tr>
<td>RSOLVE_E_READONLY</td>
<td>The set is read-only and cannot be modified.</td>
</tr>
<tr>
<td>RSOLVE_E_SETLOCKED</td>
<td>The set is locked and cannot be modified at the current time.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>
IRSetModifier

<table>
<thead>
<tr>
<th>E_FAIL</th>
<th>The operation failed due to an unspecified reason.</th>
</tr>
</thead>
</table>

This method may also return any of the error codes from the COM function `VariantChangeType`.

### 1.25.2 IRSetModifier::Clear

Removes all items from the set.

```cpp
HRESULT Clear();
```

**Return values**

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSOLVE_E_READONLY</td>
<td>The set is read-only and cannot be modified.</td>
</tr>
<tr>
<td>RSOLVE_E_SETLOCKED</td>
<td>The set is locked and cannot be modified at the current time.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

### 1.25.3 IRSetModifier::Insert

Inserts a new item into the set, optionally before a specific element in the set.

```cpp
HRESULT Insert(
    [in] VARIANT Item,
    [in, optional] VARIANT AtIndex
);
```

**Parameters**

- **Item**
  
  `[in] VARIANT` specifying the item to add to the set.

- **AtIndex**
  
  `[in, optional]` A `VARIANT` specifying at what index the item should be inserted. All items with an index greater than or equal to the specified index will have their index increased by one. See the Remarks section for more information.

**Return values**

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
</table>
This method may also return any of the error codes from the COM function VariantChangeType.

Remarks

To specify an index, set the vt member of the AtIndex parameter to VT_I4 and the IVal member to the desired index. To specify that the item should be added to the end of the set, set the vt member to VT_ERROR and the scode member to DISP_E_PARAMNOTFOUND. This is the standard way to indicate optional parameters in VARIANTs.

1.25.4 IRSetModifier::Remove

Removes an item from the set.

HRESULT Remove(
    [in] VARIANT Item
);

Parameter

Item

[in] VARIANT specifying the item to remove from the set.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_READONLY</td>
<td>The set is read-only and cannot be modified.</td>
</tr>
<tr>
<td>RSOLVE_E_SETLOCKED</td>
<td>The set is locked and cannot be modified at the current time.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The AtIndex parameter was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
IRSetModifier

| E_FAIL | The operation failed due to an unspecified reason. |

### 1.25.5 IRSetModifier::RemoveAt

Removes an item at a specific index from the set.

```c
HRESULT RemoveAt([in] long Index);
```

**Parameter**

**Index**

[in] Index of the item to remove.

**Return values**

| S_OK | The operation completed successfully. |
| RSOLVE_E_READONLY | The set is read-only and cannot be modified. |
| RSOLVE_E_SETLOCKED | The set is locked and cannot be modified at the current time. |
| E_INVALIDARG | The Index was invalid. |
| E_OUTOFMEMORY | The operation failed due to insufficient memory. |
| E_FAIL | The operation failed due to an unspecified reason. |

### 1.25.6 IRSetModifier::Set

Assigns a new value to an item at a specific index in the set.

```c
HRESULT Set([in] long Index, [in] VARIANT Item);
```

**Parameters**

**Index**

[in] Index of the item to set.

**Item**
[in] VARIANT containing the new item to place into the set at the specified index.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_READONLY</td>
<td>The set is read-only and cannot be modified.</td>
</tr>
<tr>
<td>RSOLVE_E_SETLOCKED</td>
<td>The set is locked and cannot be modified at the current time.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

This method may also return any of the error codes from the COM function VariantChangeType.

1.26 IRSolver

IRSolver is the primary solver interface, implemented by all solver objects in the framework. It provides for global control of the solver.

When to implement. Implement IRSolver on all solver objects.

When to use. Use IRSolver to control the global characteristics of a solver, including setting and retrieving its solver sites, synchronous solving, clearing inputs and outputs, and to set up advise connections with the solver.

Methods in vtable order.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IUnknown methods:</td>
<td></td>
</tr>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
<tr>
<td>IRSolver methods:</td>
<td></td>
</tr>
<tr>
<td>GetSolverSiteIn</td>
<td>Returns the inbound solver site containing this solver.</td>
</tr>
<tr>
<td>SetSolverSiteIn</td>
<td>Sets the inbound solver site containing this solver.</td>
</tr>
<tr>
<td>GetSolverSiteOut</td>
<td>Returns the outbound solver site containing this solver.</td>
</tr>
<tr>
<td>SetSolverSiteOut</td>
<td>Sets the outbound solver site containing the solver.</td>
</tr>
<tr>
<td>Solve</td>
<td>Synchronously executes the solver.</td>
</tr>
<tr>
<td>ClearInputs</td>
<td>Clears the solver’s inputs.</td>
</tr>
<tr>
<td>ClearOutputs</td>
<td>Clears the solver’s outputs.</td>
</tr>
<tr>
<td>SolveAdvise</td>
<td>Establishes an advisory connection with the solver.</td>
</tr>
<tr>
<td>SolveUnadvise</td>
<td>Destroys an advisory connection with the solver.</td>
</tr>
</tbody>
</table>
1.26.1 IRSolver::ClearInputs

Instructs the solver to clear any inputs it is managing and any other internal buffers related to the inputs. This method is usually called by the inbound solver site, although a client may call it under unusual circumstances.

HRESULT ClearInputs();

Return values

<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INPUTSLOCKED</td>
<td>The inputs were locked and could not be cleared.</td>
</tr>
<tr>
<td>RSOLVE_E_SOLVING</td>
<td>The solver was already executing. This method should not be called while the solver is executing.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.26.2 IRSolver::ClearOutputs

Instructs the solver to clear any outputs it is managing and any other internal buffers related to the outputs. This method is usually called by the outbound solver site, although a client may call it under unusual circumstances.

HRESULT ClearOutputs();

Return values

<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_OUTPUTSLOCKED</td>
<td>The outputs were locked and could not be cleared.</td>
</tr>
<tr>
<td>RSOLVE_E_SOLVING</td>
<td>The solver was already executing. This method should not be called while the solver is executing.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.26.3 IRSolver::GetSolverSiteIn

Returns the inbound solver site associated with this solver.

HRESULT GetSolverSiteIn(
    [in] REFIID riid,
)
[out, iid_is(riid)] LPVOID* ppVoid

Parameters

riid

[in] Identifier of the interface to request on the inbound solver site.

ppVoid

[out] Pointer to location that upon successful completion points to the requested interface of the inbound solver site. If the inbound solver site does not support the interface specified in iid, *ppvVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The inbound solver site does not support the requested interface.</td>
</tr>
<tr>
<td>RSOLVE_E_NOSOLVERSITE</td>
<td>The inbound solver site has not yet been set in the solver.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.26.4 IRSolver::GetSolverSiteOut

Returns the outbound solver site associated with this solver.

HRESULT GetSolverSiteOut(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);

Parameters

riid

[in] Identifier of the interface to request on the outbound solver site.

ppVoid
[out] Pointer to location that upon successful completion points to the requested interface of the outbound solver site. If the outbound solver site does not support the interface specified in iid, *ppvVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The outbound solver site does not support the requested interface.</td>
</tr>
<tr>
<td>RSOLVE_E_NOSOLVERSITE</td>
<td>The outbound solver site has not yet been set in the solver.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.26.5 IRSolver::SetSolverSiteIn

Assigns an inbound solver site to be associated with this solver.

```cpp
HRESULT SetSolverSiteIn(
    [in] LPUNKNOWN pUnk
);  
```

Parameter

pUnk

[in] Pointer to the IUnknown implementation of the inbound solver site. This value may be zero, to indicate that the solver should release its current inbound solver site if it has one.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The operation failed because the parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The solver releases its current inbound solver site, if it has one, before assigning the new one.
1.26.6 IRSolver::SetSolverSiteOut

Assigns an outbound solver site to be associated with this solver.

```c
HRESULT SetSolverSiteOut(
    [in] LPUNKNOWN pUnk
);```

**Parameter**

`pUnk`

[in] Pointer to the IUnknown implementation of the outbound solver site. This value may be zero, to indicate that the solver should release its current outbound solver site if it has one.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The operation failed because the parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

The solver releases its current outbound solver site, if it has one, before assigning the new one.

1.26.7 IRSolver::Solve

Instructs the solver to execute synchronously.

```c
HRESULT Solve();```

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_NOTALLINPUTS</td>
<td>Not all inputs were set into the solver.</td>
</tr>
<tr>
<td>RSOLVE_E_SOLVING</td>
<td>The solver was already executing. This method should not be called while the solver is executing.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>


1.26.8 IRSolver::SolveAdvise

Called by an object supporting a solver advise sink to create a connection between the solver and the advise sink. This enables the advise sink to be notified of progress and completion of the solver.

```cpp
HRESULT SolveAdvise(
    [in] IRSolverAdvise* pAdvise,
    [out] DWORD* pdwCookie
);
```

Parameters

pAdvise

[in] Pointer to the IRSolverAdvise sink that will receive solver notifications.

pdwCookie

[out] Pointer to a DWORD cookie that identifies this connection. Use this cookie later to delete the advisory connection (by passing it to IRSolver::SolveUnadvise). If this value is zero, the connection was not established.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The advisory connection was established.</td>
</tr>
<tr>
<td>OLE_E_ADVISENOTSUPPORTED</td>
<td>The solver does not support solver notifications.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The advise sink pointer was invalid.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Note to implementers

To simplify the implementation of SolveAdvise and the other notification methods in IRSolver, the framework provides an advise holder object that manages the registration and sending of notifications. To get a pointer to this object, create the SolverAdviseHolder object (CLSID CLSID_RSolverAdviseHolder) on the first invocation of SolveAdvise. This supplies a pointer to the object's IRSolverAdviseHolder interface. Then, delegate the call to the IRSolverAdviseHolder::Advise method in the data advise holder, which creates, and subsequently manages, the requested connection.
1.26.9 IRSolver::SolveUnadvise

Destroys an advisory notification connection.

```cpp
HRESULT Unadvise(
    [in] DWORD dwCookie
);
```

Parameter

dwCookie

[in] A cookie that identifies the connection to destroy. This cookie must have been previously obtained from a call to IRSolver::SolveAdvise.

Return values

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The connection was successfully terminated.</td>
</tr>
<tr>
<td>OLE_E_NOCONNECTION</td>
<td>The specified dwCookie is not a valid connection.</td>
</tr>
<tr>
<td>OLE_E_ADVISENOTSUPPORTED</td>
<td>This IRSolver implementation does not support notification.</td>
</tr>
</tbody>
</table>

Remarks

This method destroys a notification created with a call to IRSolver::SolveAdvise method.

If the advisory connection being deleted was initially set up by delegating the IRSolver::SolveAdvise call to IRSolverAdviseHolder::Advise, this call must be delegated to IRSolverAdviseHolder::Unadvise to delete it.

1.27 IRSolverAdvise

The IRSolverAdvise interface enables solver sites, mappings, clients, and other interested objects to receive solver progress and completion notifications from a solver. For an advisory connection to exist, the object that is to receive notifications must implement or contain a sub-object that implements IRSolverAdvise. An outbound solver site uses IRSolverAdvise to be notified when the solver it wraps has finished processing.

Objects interested in receiving the solver notifications should establish an advisory connection with the solver by calling IRSolver::SolveAdvise. A connection can be terminated by calling IRSolver::SolveUnadvise. While the connection is active, the solver may call the IRSolverAdvise notifications at any time.

**When to implement.** IRSolverAdvise should be implemented on those objects that are interested in receiving directly solver notifications.
When to use. Solvers use IRSolverAdvise to notify their advisory connections of their progress of completion. Normally, other objects do not need to call methods on this interface.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRSolverAdvise methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSolveComplete</td>
<td>The solver is finished processing.</td>
</tr>
<tr>
<td>OnSolveNotify</td>
<td>Notification of solver progress during processing.</td>
</tr>
</tbody>
</table>

1.27.1 IRSolverAdvise::OnSolveComplete

Called by a solver to notify its registered advise sinks that it has completed processing.

```c
HRESULT OnSolveComplete(
    [in] DWORD dwResult
);
```

Parameter

dwResult

[in] A DWORD flag indicating the result of the solver's execution. It can take one of the values from the RSOLVE_SOLVE_COMPLETE_RESULTS enumeration.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

Generally, the solver can ignore any of the return values because this is a notification method.

1.27.2 IRSolverAdvise::OnSolveNotify

Called by a solver to notify its registered advise sinks of its progress during execution.

```c
HRESULT OnSolveNotify(
    [in] DWORD dwFlag,
    [in] double dPercComplete,
    [in] long lHoursLeft,
    [in] long lMinutesLeft,
```
[in] long lMillisecondsLeft,
[in, out] DWORD* pfAction
);

Parameters

dwFlag

[in] A DWORD bitmask indicating which of the remaining input parameters are valid. Its values can come from the RSOLVE_SOLVE_NOTIFY_FLAGS enumeration.

dPercComplete

[in] A double indicating the percentage processing complete, as a value between 0.0 and 1.0. This item is valid if the dwFlag parameter has the RSNF_PERCENTAGE bit set; otherwise, the parameter has no meaning.

lHoursLeft

[in] A long indicating the estimated number of hours remaining, if the dwFlag parameter has the RSNF_HOURS bit set; otherwise, the parameter has no meaning.

lMinutesLeft

[in] A long indicating the estimated number of minutes remaining, if the dwFlag parameter has the RSNF_MINUTES bit set; otherwise, the parameter has no meaning.

lMillisecondsLeft

[in] A long indicating the estimated number of milliseconds remaining, if the dwFlag parameter has the RSNF_MILLISECONDS bit set; otherwise, the parameter has no meaning.

pfAction

[in, out] Pointer to a DWORD that upon return indicates the action the solver should take. The action can take values from the enumeration RSOLVE_SOLVE_NOTIFY_ACTIONS. The caller should initialize this value to the default action, RSNA_CONTINUE.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
**Remarks**

The total estimated time remaining is the sum of the valid time parameters `lHoursLeft`, `lMinutesLeft`, and `lMillisecondsLeft`.

**Note to implementers**

Leave the `pfAction` parameter unchanged to accept the current action, which may have been set by a previous advise sink in the notification chain.

### 1.28 IRSolverAdviseHolder

The `IRSolverAdviseHolder` interface contains methods that create and manage advisory connections between a solver and one or more advise sinks. Its methods are intended to be used to implement the advisory methods of `IRSolver`. `IRSolverAdviseHolder` is implemented on an advise holder object. Its methods establish and delete solver advisory connections and send notification of progress and completion of the solver to an object that requires this notification, such as an outbound solver site, which must contain an advise sink.

Advise sinks are objects that require notification of progress or completion of the solver and implement the `IRSolverAdvise` interface.

**When to implement.** Typically, use the framework-provided implementation of `IRSolverAdviseHolder` to simplify the implementation of the `SolveAdvise` and `SolveUnadvise` methods in the `IRSolver` interface, and to send notifications as appropriate. It would be necessary to implement `IRSolverAdviseHolder` only in the case where there may be a need for a custom solver advise holder object, whose methods are to be used to implement the `IRSolver` methods in a set of servers.

**When to use.** The implementation of the advisory methods of `IRSolver` can call the methods in `IRSolverAdviseHolder`. The first time the solver receives a call to `IRSolver::SolveAdvise`, create an object of `CLSID_CLSID_RSolverAdviseHolder` to create an instance of the framework-provided advise holder and get a pointer to its `IRSolverAdviseHolder` interface. Then, in implementing the `IRSolver` interface on the solver, delegate implementations of the `SolveAdvise` and `SolveUnadvise` methods to the corresponding methods in `IRSolverAdviseHolder`.

When interesting progress or completion occurs, call `IRSolverAdviseHolder::SendOnSolveNotify` or `IRSolverAdviseHolder::SendOnSolveComplete` from the solver to carry out the necessary notifications.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IUnknown</code> methods:</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td><code>QueryInterface</code></td>
<td>Increments reference count.</td>
</tr>
<tr>
<td><code>AddRef</code></td>
<td>Decrements reference count.</td>
</tr>
<tr>
<td><code>Release</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>IRSolverAdviseHolder</code> methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Advise          Establishes advisory connection with sink.
Unadvise        Destroys advisory connection with sink.
SendOnSolveComplete  Advises sinks of solver completion.
SendOnSolveNotify        Advises sinks of solver progress.

1.28.1 IRSolverAdviseHolder::Advise

Creates an advisory connection between an advise sink and a solver for receiving notifications. This is typically called in the implementation of IRSolver::SolveAdvise.

```cpp
HRESULT Advise(
    [in] LPRSOLVER pSource,
    [in] LPRSOLVERADVISE pAdvise,
    [out] DWORD* pdwCookie
);
```

Parameters

- **pSource**
  - [in] Pointer to the solver that will provide notifications.
- **pAdvise**
  - [in] Pointer to the advise sink that will receive notifications.
- **pdwCookie**
  - [out] Pointer to a DWORD that upon return will contain a cookie that identifies this advisory connection. The client can later destroy this connection by passing this cookie to IRSolverAdviseHolder::Unadvise. If this value is zero, the connection was not established.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The connection was established.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>Either the solver or advise sink pointers was invalid.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The DWORD cookie pointer was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The solver typically calls this method in the IRSolver::SolveAdvise method, to manage the list of advise connections.
**1.28.2 IRSolverAdviseHolder::SendOnSolveComplete**

Sends notifications to each advise sink for which there is a connection established by calling the IRSolverAdvise::OnSolveComplete method for each advise sink currently being handled by this instance of the solver advise holder object.

```c
HRESULT SendOnSolveComplete(
    [in] DWORD dwResult
);
```

**Parameter**

*dwResult*

[in] A DWORD flag indicating the result of the solver's execution. It can take one of the values from the RSOLVE_SOLVE_COMPLETE_RESULTS enumeration.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

Generally, the solver can ignore any of the return values because this is a notification method.

**1.28.3 IRSolverAdviseHolder::SendOnSolveNotify**

Sends notifications to each advise sink for which there is a connection established by calling the IRSolverAdvise::OnSolveNotify method for each advise sink currently being handled by this instance of the solver advise holder object.

```c
HRESULT SendOnSolveNotify(
    [in] DWORD dwFlag,
    [in] double dPercComplete,
    [in] long lHoursLeft,
    [in] long lMinutesLeft,
    [in] long lMillisecondsLeft,
    [out] DWORD* pfAction
);
```

**Parameters**

*dwFlag*
[in] A DWORD bitmask indicating which of the remaining input parameters are valid. Its values can come from the RSOLVE_SOLVE_NOTIFY_FLAGS enumeration.

dPercComplete

[in] A double indicating the percentage processing complete, as a value between 0.0 and 1.0. This item is valid if the dwFlag parameter has the RSNF_PERCENTAGE bit set; otherwise, the parameter has no meaning.

lHoursLeft

[in] A long indicating the estimated number of hours remaining, if the dwFlag parameter has the RSNF_HOURS bit set; otherwise, the parameter has no meaning.

lMinutesLeft

[in] A long indicating the estimated number of minutes remaining, if the dwFlag parameter has the RSNF_MINUTES bit set; otherwise, the parameter has no meaning.

lMillisecondsLeft

[in] A long indicating the estimated number of milliseconds remaining, if the dwFlag parameter has the RSNF_MILLISECONDS bit set; otherwise, the parameter has no meaning.

pfAction

[out] Pointer to a DWORD that upon return indicates the action the solver should take. The action can take values from the enumeration RSOLVE_SOLVE_NOTIFY_ACTIONS.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The total estimated time remaining is the sum of the valid time parameters lHoursLeft, lMinutesLeft, and lMillisecondsLeft.

1.28.4 IRSolverAdviseHolder::Unadvise

Deletes an existing advisory connection between an advise sink and a solver that had previously been created by a call to IRSolverAdviseHolder::Advise. This is typically called in the implementation of IRSolver::SolveUnadvise.
HRSLirUnadvise{
    [in] DWORD dwCookie
};

Parameter
dwCookie

[in] A DWORD cookie that identifies the connection to delete. This cookie must have been obtained
from a prior call to IRslverAdviseHolder::Advise.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The specified connection was successfully deleted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_E_NOCONNECTION</td>
<td>The specified dwCookie was not a valid connection.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.29 IRSolverAsynchSolve

Interface IRSolverAsynchSolve enables a client or inbound solver site to execute a solver
asynchronously, if supported by the solver.

When to implement. Implement this interface on a solver if it supports asynchronous execution.
Typically, solvers that implement this interface will also implement IRSolverControl and
IRslverStatus. Note that asynchronous execution usually requires multiple-threaded programming.

When to use. Use this interface to control a solver asynchronously.

Methods in vtable order

IUnknown methods:
QueryInterface Description
AddRef Returns pointers to supported interfaces.
Release Increments reference count.
IRslverAsynchSolve methods:
AsynchSolve Decrements reference count.
Executes the solver asynchronously.

1.29.1 IRSolverAsynchSolve::AsynchSolve

Executes the solver asynchronously. The solver returns as soon as it can, and provides completion
notifications via the notification interface IRslverAdvise.

HRESULT AsynchSolve();
Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_NOTALLINPUTS</td>
<td>Not all inputs were set into the solver.</td>
</tr>
<tr>
<td>RSOLVE_E_SOLVING</td>
<td>The solver was already executing. This method should not be called while the solver is executing.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.30 IRSolverBaselnfo

IRSolverBaselnfo is the base interface from which all of the introspection interfaces derive. Every introspection object supports the properties within IRSolverBaselnfo, which include the name of the object, a short description, object-specific flags, and help information.

When to implement. Implement this interface on any per-solver implementation of the SolverInfo objects. Solvers that use the generic implementation of SolverInfo do not need to implement this interface.

When to use. Because the other SolverInfo interfaces derive from IRSolverBaselnfo, it is not necessary to use this interface specifically, unless the type of the SolverInfo object is not known.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
<tr>
<td>IRSolverBaselnfo methods:</td>
<td>Returns the name of the object.</td>
</tr>
<tr>
<td>GetName</td>
<td>Returns a short description of the object.</td>
</tr>
<tr>
<td>GetShortDescription</td>
<td>Returns flags about the object.</td>
</tr>
<tr>
<td>GetFlags</td>
<td>Returns name of help file describing object.</td>
</tr>
<tr>
<td>GetHelpFile</td>
<td>Returns help context within help file that describes object.</td>
</tr>
</tbody>
</table>

1.30.1 IRSolverBaselnfo::GetFlags

Returns flags describing boolean or enumerated properties of the information object.

```c
HRESULT GetFlags(
    [out] UINT* puFlags
);
```
IRSolverBaselnfo

Parameter

puFlags

[out] Pointer to a UINT that on successful completion contains the bitmask flags that describe the information object. See Remarks for more information.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

There is a common set of bitmask flags for all information objects, as specified by the enumeration RSOLVE_SOLVER_INFO_FLAGS. Furthermore, each information object type specifies its own set of flags that extended the RSOLVE_SOLVER_INFO_FLAGS enumeration. Currently, only the primary SolverInfo object adds flags to this bitmask value.

The primary SolverInfo object adds the enumeration RSI_SOLVERFLAGS.

1.30.2 IRSolverBaselnfo::GetHelpContext

Returns a help context identifier that, in association with a help file returned from GetHelpFile, identifies a WinHelp topic describing the solver element.

HRESULT GetHelpContext(
    [out] DWORD* pdwHelpCtx
);

Parameter

pdwHelpCtx

[out] Pointer to a DWORD which upon successful completion contains the help context identifier. Upon failure, *pdwHelpCtx is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The help context was returned correctly.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

82
1.30.3 IRSolverBaseInfo::GetHelpFile

Returns the path and name of a help file that contains a topic, identified by the help context returned from GetHelpContext, that describes the particular SolverInfo object.

```c
HRESULT GetHelpFile(
    [out, string] LPOLESTR* ppwszHelpFile
);
```

**Parameter**

`ppwszHelpFile`

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the help file path and name. Upon failure, *ppwszHelpFile* is set to zero.

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The help file was returned correctly.</td>
</tr>
<tr>
<td>S_FALSE</td>
<td>The object does not provide a help file for this particular item. *ppwszHelpFile should be set to zero.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

This method allocates memory for the string returned in the `ppwszHelpFile` parameter using the `IMalloc::Alloc` method. The caller is responsible for calling the `IMalloc::Free` method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.30.4 IRSolverBaseInfo::GetName

Returns the name of the solver element.

```c
HRESULT GetName(
    [out, string] LPOLESTR* ppwszName
);
```
IRSolverBaseInfo

Parameter

ppwszName

[out] Points to a location that upon successful completion points to a zero-terminated string containing the name of the solver element. Upon failure, *ppwszName is set to zero.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This method allocates memory for the string returned in the ppwszName parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.30.5 IRSolverBaseInfo::GetShortDescription

Returns a short description of the solver element.

HRESULT GetShortDescription(
    [out, string] LPOLESTR* ppwszDescription
);

Parameter

ppwszDescription

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the short description of the solver element. Upon failure, *ppwszDescription is set to zero.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
</tbody>
</table>
Remarks

This method allocates memory for the string returned in the ppwszDescription parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.31 IRSolverControl

IRSolverControl enables any client to asynchronously control the execution of a solver. Interfaces IRSolverControl and IRSolverStatus are the only two interfaces that, if supported, can be called at any time by any client. Solvers that can be queried asynchronously typically register themselves in the running object table (ROT). Clients can then acquire pointers to the solvers through the ROT, and then control them using the IRSolverControl interface.

When to implement. Implement this interface on a solver if the solver supports life cycle control from any client at any time. Usually, this interface is implemented on a separate thread so that it can receive messages at any time.

When to use. This interface is typically used by the primary client of a solution application or by clients that have no prior association with the solver, but need to connect with the solver to control it, such as in an emergency shut-down situation.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRSolverControl methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort</td>
<td>Aborts solver execution.</td>
</tr>
<tr>
<td>Pause</td>
<td>Pauses solver execution.</td>
</tr>
<tr>
<td>Resume</td>
<td>Resumes paused solver execution.</td>
</tr>
<tr>
<td>Wait</td>
<td>Pauses solve execution for a fixed time.</td>
</tr>
</tbody>
</table>

1.31.1 IRSolverControl::Abort

Instructs the solver to halt its processing and discard any internal states as soon as possible.

HRESULT Abort();

Return values

| S_OK                                        | The operation completed successfully. |
IRSolverControl

E_FAIL The operation failed due to an unspecified reason.

Remarks
Because IRSolverControl is essentially an asynchronous mechanism into the solver, the solver may not have halted processing when this function returns. The solver will send out its OnSolveComplete notification after it has finished.

1.31.2 IRSolverControl::Pause
Instructs the solver to pause (halt) its processing, while retaining its internal state.

HRESULT Pause();

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks
Because IRSolverControl is essentially an asynchronous mechanism into the solver, the solver may not have paused processing when this function returns.

1.31.3 IRSolverControl::Resume
Instructs a paused solver to resume its processing.

HRESULT Resume();

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_UNEXPECTED</td>
<td>The solver was not paused.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.31.4 IRSolverControl::Wait
Instructs a solver to pause for a specific time period before resuming its processing.

HRESULT Wait(
    [in] UINT uMilliseconds
);
Parameter

uMilliseconds

[in] Number of milliseconds the solver should pause its processing.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.32 IRSolverDescription

The IRSolverDescription interface, part of the SolverDescription introspection specification, enables a client to determine qualitative capabilities of a solver, such as the author, developer, runtime, and URL for the solver.

When to implement. The generic SolverInfo implementation object provided by the core services implements this interface for those solvers that use the SolverInfo Type Library to describe their SolverInfo and SolverDescription properties. It is necessary to implement this interface only for those solvers that implement their own SolverDescription object, as provided by IRSolverProvideInfo.

When to use. A client uses IRSolverDescription to determine the properties described by its methods, usually in the process of displaying these properties to the user in a solver browser.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

IRSolverDescription methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAuthor</td>
<td>Returns solver’s author.</td>
</tr>
<tr>
<td>GetDeveloper</td>
<td>Returns solver’s developer.</td>
</tr>
<tr>
<td>GetCitation</td>
<td>Returns solver’s citation.</td>
</tr>
<tr>
<td>GetSolutionFlags</td>
<td>Returns flags relating to solver’s capabilities.</td>
</tr>
<tr>
<td>GetWorstRuntime</td>
<td>Returns solver’s worst runtime as text string.</td>
</tr>
<tr>
<td>GetWorstRuntimeAsTeX</td>
<td>Returns solver’s worst runtime as a TeX string.</td>
</tr>
<tr>
<td>GetURL</td>
<td>Returns the URL for the solver’s web site.</td>
</tr>
</tbody>
</table>

1.32.1 IRSolverDescription::GetAuthor

Returns the author of the algorithm implemented by the solver.

HRESULT GetAuthor(
    [out, string] LPOLESTR* ppwszAuthor

IRSolverDescription

};

Parameter

ppwszAuthor

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the author description property of the solver. Upon failure, *ppwszAuthor is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The author description property was returned correctly.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

If this description property is undefined for the solver, this will return an allocated empty string.

This method allocates memory for the string returned in the ppwszAuthor parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.32.2 IRSolverDescription::GetCitation

Returns a bibliographic citation to an article or other reference of the algorithm implemented by the solver.

HRESULT GetCitation(
    [out, string] LPOLESTR* ppwszCitation
);

Parameter

ppwszCitation

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the citation description property of the solver. Upon failure, *ppwszCitation is set to zero.
Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The citation description property was returned correctly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

If this description property is undefined for the solver, this will return an allocated empty string.

This method allocates memory for the string returned in the ppwszCitation parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.32.3 IRSolverDescription::GetDeveloper

Returns the developer or developer’s company that implemented the solver.

```c
HRESULT GetDeveloper(
    [out, string] LPOLESTR* ppwszDeveloper
);```

Parameter

`ppwszDeveloper`

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the developer description property of the solver. Upon failure, *ppwszDeveloper is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The developer description property was returned correctly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
IRSolverDescription

Remarks

If this description property is undefined for the solver, this will return an allocated empty string.

This method allocates memory for the string returned in the ppwszDeveloper parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.32.4 IRSolverDescription::GetSolutionFlags

Returns a bitmask indicating capabilities of the solver with regard to the solutions it generates.

HRESULT GetSolutionFlags(
    [out] DWORD* pdwFlags
);

Parameter

pdwFlags

[out] A pointer to a DWORD value that upon successful completion contains flags from the RSD_DESCRIPTIONFLAGS enumeration. Upon failure, \*pdwFlags is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The solution flags were successfully returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.32.5 IRSolverDescription::GetURL

Returns a URL pointing to a web site or local resource that provides information about the solver.

HRESULT GetURL(
    [out, string] LPOLESTR* ppwszURL
);

Parameter

ppwszURL

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the URL description property of the solver. Upon failure, \*ppwszURL is set to zero.
Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The URL description property was returned correctly.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

If this description property is undefined for the solver, this will return an allocated empty string.

This method allocates memory for the string returned in the ppwszURL parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.32.6 IRSolverDescription::GetWorstRuntime

Returns the asymptotically worst-case runtime of the solver, as a readable text string.

```c
HRESULT GetWorstRuntime(  
    [out, string] LPOLESTR* ppwszRuntime  
);
```

Parameter

ppwszRuntime

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the worst-case runtime description property of the solver. Upon failure, *ppwszRuntime is set to zero.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The worst-case runtime description property was returned correctly.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
IRSolverDescription

Remarks

If this description property is undefined for the solver, this will return an allocated empty string.

This method allocates memory for the string returned in the ppwszRuntime parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.32.7 IRSolverDescription::GetWorstRuntimeAsTeX

Returns the asymptotically worst-case runtime of the solver, as a TeX string that could be displayed graphically as a mathematical equation.

HRESULT GetWorstRuntimeAsTeX(
    [out, string] LPOLESTR* ppwszTeX
);

Parameter

ppwszTeX

[out] Pointer to a location that upon successful completion points to a zero-terminated string containing the worst-case runtime description property of the solver as a TeX string. Upon failure, *ppwszTeX is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The worst runtime as TeX description property was returned correctly.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

If this description property is undefined for the solver, this will return an allocated empty string.

This method allocates memory for the string returned in the ppwszTeX parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.
1.33 IRSolverDescriptionProperties

IRSolverDescriptionProperties enables a client to access all of the description properties of a solver without knowing \textit{a priori} the names or description property identifiers of the properties supported by the solver.

\textbf{When to implement.} The generic SolverInfo implementation object provided by the core services implements this interface for those solvers that use the SolverInfo Type Library to describe their SolverInfo and SolverDescription properties. It is necessary to implement this interface only for those solvers that implement their own SolverDescription object, as provided by IRSolverProvideInfo.

\textbf{When to use.} A client uses IRSolverDescriptionProperties to determine the description properties supported by solver when the client does not know the supported properties in advance, or when the solver supports properties beyond those defined in IRSolverDescription, usually in the process of displaying these properties to the user in a solver browser.

\textbf{Methods in vtable order}

}\texttt{IUnknown methods:} Description
\begin{itemize}
\item QueryInterface: Returns pointers to supported interfaces.
\item AddRef: Increments reference count.
\item Release: Decrements reference count.
\end{itemize}

\texttt{IRSolverDescriptionProperties methods:}
\begin{itemize}
\item LookupProperty: Looks up a description property of a solver.
\item EnumDescriptionProperties: Enumerates all description properties of a solver.
\end{itemize}

1.33.1 IRSolverDescriptionProperties::EnumDescriptionProperties

Returns an enumerator object that enumerates over all of the description properties supported by the solver.

\begin{verbatim}
HRESULT EnumDescriptionProperties(
    [out] IEnumRDESCRIPTIONPROPERTY** ppenumDescProp
);
\end{verbatim}

\textbf{Parameter}

\begin{itemize}
\item ppenumDescProp [out] Pointer to location that upon successful completion points to the IEnumRDESCRIPTIONPROPERTY interface that enumerates the description properties for the solver.
\end{itemize}

\textbf{Return values}

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>
IRSolverDimInfo

<table>
<thead>
<tr>
<th>E_POINTER</th>
<th>The output parameter was an invalid pointer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.33.2 IRSolverDescriptionProperties::LookupProperty

Given a description property identifier, returns the requested description property of the solver, if supported by the solver.

```c
HRESULT LookupProperty(
    [in] REFRDESCPROPID rdpid,
    [out] VARIANT* pVarProp
);
```

**Parameters**

- **rdpid**
  
  [in] The description property identifier of the description property to look up on the solver.

- **pVarProp**
  
  [out] Pointer to a caller-allocated VARIANT that upon successful completion contains the requested description property.

**Return values**

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_INVALIDARG</td>
<td>The requested description property was invalid or was not supported by the solver.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.34 IRSolverDimInfo

Interface IRSolverDimInfo, part of the introspection specification, describes the properties of a dimension object associated with an input or an output of a solver.

**When to implement.** Implement this interface on per-solver, custom implementations of the dimension SolverInfo object. Solvers that use the generic implementation of SolverInfo do not need to implement this interface.
**When to use.** Use this interface to determine dimension properties of a dimension object associated with an input or output of a solver. In particular, it provides access to the set associated with the dimension.

**Methods in vtable order**

*IUnknown methods:*

- QueryInterface
- AddRef
- Release

*IRSolverBaselnfo methods:*

- GetName
- GetShortDescription
- GetFlags
- GetHelpFile
- GetHelpContext

*IRSolverDimInfo method:

- GetSetIndex

Returns index of the set that describes this dimension.

### 1.34.1 IRSolverDimInfo::GetSetIndex

Returns the index of the set information object within the primary SolverInfo object that describes the range of this dimension.

```c
HRESULT GetSetIndex(
    [out] UINT* plndex
);
```

**Parameter**

*plndex*

[out] Pointer to a UINT that upon successful completion contains the index of the set that characterizes the meaning and range of this dimension.

**Return values**

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

The set information object characterizing this dimension can be obtained by calling IRSolverInfo::GetSetInfo and passing it the value returned from this method.
1.35 IRSolverEnumeration

Interface IRSolverEnumeration enables a client to enumerate the solvers installed on the system. This interface is part of the core services.

**When to implement.** Generally, it is not necessary to implement this interface. The core services implementation of the SolverRegistrar implements this interface.

**When to use.** Use this interface to enumerate the list of solvers installed on the system. This is useful in a browser of solvers for the user, for instance. The core services implementation of this interface can be acquired by creating the SolverRegistrar object, which has CLSID CLSID_RSolverRegistrar.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

**IRSolverEnumeration methods:**

| EnumSolvers               | Enumerates the solvers on the system.            |
| EnumSolversWithInfo       | Enumerates the solvers and their SolverInfos on the system. |

1.35.1 IRSolverEnumeration::EnumSolvers

Returns an enumerator objects that enumerates the CLSIDs of the solvers installed on the system.

```c
HRESULT EnumSolvers(
    [out] IEnumCLSID** ppenumCLSID
);
```

**Parameter**

`ppenumCLSID`

[out] Pointer to location that upon successful completion contains a pointer to the IEnumCLSID interface, which enumerates the CLSIDs of the solvers installed on the system. Upon failure, *ppenumCLSID is set to zero.

**Return values**

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
1.35.2 IRSolverEnumeration::EnumSolversWithInfo

Enumerates the CLSIDs of the solvers and the GUIDs of the SolverInfos of the solvers installed on the system.

```c
HRESULT EnumSolversWithInfo(
    [out] IEnumRREGSOLVERINFO** ppenumSolverInfo
);
```

**Parameter**

`ppenumSolverInfo`

[out]Pointer to location that upon successful completion contains a pointer to the `IEnumRREGSOLVERINFO` interface, which enumerates the CLSIDs of the solvers and the GUIDs of the SolverInfo objects for solvers installed on the system. Upon failure, `*ppenumSolverInfo` is set to zero.

**Return values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.36 IRSolverInfo

Interface `IRSolverInfo`, part of the introspection specification, describes the properties of a solver. This is the primary interface implemented by the primary, first SolverInfo object returned on a solver.

**When to implement.** Implement this interface on per-solver, custom implementations of the main SolverInfo object. Solvers that use the generic implementation of SolverInfo do not need to implement this interface.

**When to use.** Use this interface to determine basic characteristics of a solver, or to access any of the information for inputs, outputs, parameters, or sets of a solver.

**Methods in vtable order**

- **IUnknown methods:**
  - `QueryInterface`: Returns pointers to supported interfaces.
  - `AddRef`: Increments reference count.
  - `Release`: Decrements reference count.

- **IRSolverBaseInfo methods:**
IRSolverInfo

IRSolverInfo methods:

- GetName
- GetShortDescription
- GetFlags
- GetHelpFile
- GetHelpContext
- GetInputInfoCount
- GetInputInfo
- GetOutputInfoCount
- GetOutputInfo
- GetSetInfoCount
- GetSetInfo
- GetParamInfoCount
- GetParamInfo

Returns the name of the object.

Returns a short description of the object.

Returns flags about the object.

Returns name of help file describing object.

Returns help context within help file that describes object.

Returns number of solver inputs.

Returns information object on a solver input.

Returns number of solver outputs.

Returns information object on a solver output.

Returns number of solver sets.

Returns information object on a solver set.

Returns number of solver parameters.

Returns information object on a solver parameter.

1.36.1 IRSolverInfo::GetInputInfo

Returns the input information for a specified input of the solver.

HRESULT GetInputInfo(
    [in] UINT index,
    [out, retval] LPRSOLVERINPUTINFO* pplnputlnfo
);

Parameters

index

[in] Index of the input to retrieve the input information for. This value must be between zero and one less than the value returned from IRSolverInfo::GetInputInfoCount.

pplnputlnfo

[out] Pointer to a location that on successful completion contains the IRSolverInputInfo pointer to the requested input information object. Upon failure, *pplnputlnfo is set to zero.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The input index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

98
1.36.2 IRSolverInfo::GetInputInfoCount

Returns the total number of inputs (and hence input information objects) available for the solver. This includes required and optional inputs.

HRESULT GetInputInfoCount(
    [out] UINT* pulnputs
);

Parameter

pulnputs

[out] Pointer to a UINT that upon successful completion contains the number of inputs provided by the solver. On failure, or if the solver has no inputs, *pulnputs is set to zero.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.36.3 IRSolverInfo::GetOutputInfo

Returns the output information for a specified output of the solver.

HRESULT GetOutputInfo(
    [in] UINT index,
    [out, retval] LPRSOLVEROUTPUTINFO* ppOutputInfo
);

Parameters

index

[in] Index of the output to retrieve the output information for. This value must be between zero and one less than the value returned from IRSolverInfo::GetInputInfoCount.

ppOutputInfo

[out] Pointer to a location that on successful completion will contain the IRSolverOutputInfo pointer to the requested output information object. Upon failure, *ppOutputInfo is set to zero.
IRSolverInfo

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The output index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.36.4 IRSolverInfo::GetOutputInfoCount

Returns the total number of outputs (and hence output information objects) available for the solver.

```c
HRESULT GetOutputInfoCount(
    [out] UINT* puOutputs
);
```

Parameter

puOutputs

[out] Pointer to a UINT that upon successful completion contains the number of outputs provided by the solver. On failure, or if the solver has no outputs, *puOutputs is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.36.5 IRSolverInfo::GetParamInfo

Returns the parameter information for a specified parameter of the solver.

```c
HRESULT GetParamInfo(
    [in] UINT index,
    [out, retval] LPRSOLVERPARAMINFO* ppParamInfo
);
```

Parameters

index
[in] Index of the parameter to retrieve the parameter information for. This value must be between zero and one less than the value returned from IRSolverInfo::GetParamInfoCount.

ppParamInfo

[out] Pointer to a location that on successful completion contains the IRSolverParamInfo pointer to the requested parameter information object. Upon failure, *ppParamInfo is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The parameter index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.36.6 IRSolverInfo::GetParamInfoCount

Returns the total number of parameters (and hence parameter information objects) available for the solver.

    HRESULT GetParamInfoCount(
        [out] UINT* puParams
    );

Parameter

puParams

[out] Pointer to a UINT that upon successful completion contains the number of parameters provided by the solver. On failure, or if the solver has no parameters, *puParams is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.36.7 IRSolverInfo::GetSetInfo

Returns the set information for a specified set of the solver.
IRSolverlnfo

HRESULT GetSetInfo(
    [in] UINT index,
    [out, retval] LPRSOLVERSETINFO* ppSetInfo
);

Parameters

index

[in] Index of the set to retrieve the set information for. This value must be between zero and one less than the value returned from IRSolverlnfo::GetSetInfoCount.

ppSetInfo

[out] Pointer to a location that on successful completion will contain the IRSolverSetInfo pointer to the requested set information object. Upon failure, *ppSetInfo is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The set index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.36.8 IRSolverlnfo::GetSetInfoCount

Returns the total number of sets (and hence set information objects) available for the solver.

HRESULT GetSetInfoCount(
    [out] UINT* puSets
);

Parameter

puSets

[out] Pointer to a UINT that upon successful completion contains the number of sets provided by the solver. On failure, or if the solver has no sets, *puSets is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
</table>
Interface IRSolverInputlnfo provides information about an input required by the solver. It is implemented within the context of SolverInfo objects for a solver.

When to implement. Implement this interface on sub-objects of the primary SolverInfo object that provide information about inputs to a solver. It is necessary to implement this interface on these sub-objects for solvers that provide their own implementation of the SolverInfo protocol. The generic SolverInfo implementation provides an implementation of this interface, and for solvers that use the generic SolverInfo implementation, it is not necessary to implement this interface.

When to use. Clients use this interface to discover properties of an input required by a solver. In particular, this interface is useful for displaying these properties to the user through a solver browser.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

| IRSolverBaseInfo methods: | |
|---------------------------| Description |
| GetName                   | Returns the name of the object. |
| GetShortDescription       | Returns a short description of the object. |
| GetFlags                  | Returns flags about the object. |
| GetHelpFile               | Returns name of help file describing object. |
| GetHelpContext            | Returns help context within help file that describes object. |

| IRSolverInputInfo methods: | |
|---------------------------| Description |
| GetDimlnfoCount           | Returns number of dimension of the input. |
| GetDimInfo                | Returns a dimension information object. |
| GetAssignmentFlags        | Returns flags concerning assignment capabilities. |
| GetChangeFlags            | Returns flags concerning change capabilities. |
| GetDataType               | Returns data type required for the input. |

1.37.1 IRSolverInputInfo::GetAssignmentFlags

Returns flags describing the assignment capabilities and requirements of an input to a solver.

```c
HRESULT GetAssignmentFlags(
    [out] DWORD* pFlags
);
```
IRSolverInputInfo

Parameter

pFlags

[out] Pointer to a DWORD that upon successful completion contains the assignment flags. These values are from the enumeration RSOLVE_SOLVER_ASSIGNMENT_FLAGS. Upon failure, *pFlags is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.37.2 IRSolverInputInfo::GetChangeFlags

Returns flags describing the modification capabilities of an input to a solver.

```c
HRESULT GetChangeFlags(
    [out] DWORD* pFlags
);
```

Parameter

pFlags

[out] Pointer to a DWORD that upon successful completion contains the change flags. These values are from the enumeration RSOLVE_SOLVER_INPUT_CHANGE_FLAGS. Upon failure, *pFlags is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.37.3 IRSolverInputInfo::GetDataType

Returns the data type of the input.

```c
HRESULT GetDataType(
    [out] LPRSOLVER_DATATYPE pType
);
```
Parameter

pType

[out] Pointer to a RSOLVER_DATATYPE structure which upon successful completion contains the data type of the input.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The RSOLVER_DATATYPE typedefs to VARTYPE, the type field for VARIANT structures.

1.37.4 IRSolverInputInfo::GetDimInfo

Returns the dimension information for a specified dimension of the input.

HRESULT GetDimInfo(
    [in] UINT index,
    [out, retval] LPRSOLVERDIMINFO* ppDimInfo
);

Parameters

index

[in] Index of the dimension to retrieve the dimension information for. This value must be between zero and one less than the value returned from IRSolverInputInfo::GetDimInfoCount.

ppDimInfo

[out] Pointer to a location which on successful completion contains the IRSolverDimInfo pointer to the requested dimension information object. Upon failure, *ppDimInfo is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The dimension index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>
1.37.5 IRSolverInputInfo::GetDimInfoCount

Returns the number of dimensions required by this input.

HRESULT GetDimInfoCount(
    [out] UINT* puDims
);

Parameter

puDims

[out] Pointer to a UINT that upon successful completion contains the number of dimensions the data element must have for this input. Upon failure, or if the input is a scalar, *puDims is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

Scalar values should set *puDims to zero, not return an error.

1.38 IRSolverInputs

IRSolverInputs is the primary solver interface for managing, accessing, and setting the inputs of a solver, implemented by all solver objects in the framework.

When to implement. Implement IRSolverInputs on all solver objects and inbound solver sites. Inbound solver sites use IRSolverInputs to wrap the input side of a solver.

When to use. Use IRSolverInputs to control the inputs of a solver, including setting and retrieving its inputs, and locking and unlocking the inputs.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
</tbody>
</table>
IRSolverInputs

IRSolverInputs methods:

- LockInputs
- UnlockInputs
- InputsLocked
- GetInputCount
- GetInputData
- SetInputData

- Release
  - Decrements reference count.
- LockInputs
  - Locks the inputs.
- UnlockInputs
  - Unlocks the inputs.
- InputsLocked
  - Returns whether the inputs are locked.
- GetInputCount
  - Returns the number of inputs.
- GetInputData
  - Returns an assigned input data element.
- SetInputData
  - Sets input data into the solver.

1.38.1 IRSolverInputs::GetInputCount

Returns the total number of inputs available for the solver. This includes required and optional inputs.

```c
HRESULT GetInputCount(
    [out] UINT* pulnputs
);
```

Parameter

- `pulnputs`
  - [out] Pointer to a UINT that upon successful completion contains the number of inputs provided by the solver. On failure, or if the solver has no inputs, `*pulnputs` is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.38.2 IRSolverInputs::GetInputData

Retrieves the data source and the index of a data element within that data source that correspond to an input on the solver.

```c
HRESULT GetInputData(
    [in] UINT ulnput,
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid,
    [out] UINT* puElement
);
```
IRSolverlnputs

Parameters

ulnput

[in] The index of the input to retrieve the data source and data element index for. This value must be
between zero and one less than the value returned from IRSolverlnputs::GetlnputCount.

riid

[in] Identifier of the interface to return on the data source that manages the desired input data
element.

ppVoid

[out] Pointer to a location that upon successful completion contains the requested interface pointer
to the data source that manages the desired input data element. Upon failure, * ppVoid is set to zero.

puElement

[out] Pointer to a UINT that upon successful completion contains the index of the data element
within the data source that corresponds to the desired output.

Return values

<table>
<thead>
<tr>
<th>Enumerated Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The index ulnput is invalid.</td>
</tr>
<tr>
<td>RSOLVE_E_INPUTNOTSET</td>
<td>The desired input was never set with a call to SetlnputData.</td>
</tr>
<tr>
<td>RSOLVE_E_CANNOTGETINPUTDATA</td>
<td>It is not possible to retrieve the desired input element as a data element. This is usually because the solver stores it in an internal format that is not compatible with the framework.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The data source does not support the requested interface.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>One of the output parameters was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

A data element is contained within a data source, and is identified within the data source by its index.
Each solver input is associated with a single data element, so in order to provide complete
information for accessing an input data element, this method returns both the data source and the index within that data source that identifies the data element.

### 1.38.3 IRSolverInputs::InputsLocked

Returns whether the inputs are currently locked or not.

```cpp
HRESULT InputsLocked(
    [out] DWORD* pfLocked
);
```

**Parameter**

`pfLocked`

[out] Pointer to a DWORD that upon successful completion is zero if the inputs are not locked, and non-zero if the inputs are locked. This value is set to zero if the operation fails.

**Return values**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

The solver maintains a (non-negative) lock count on its inputs, which is incremented by calls to `IRSolverInputs::LockInputs` and decremented by calls to `IRSolverInputs::UnlockInputs`. The inputs are locked if the lock count is non-zero.

### 1.38.4 IRSolverInputs::LockInputs

Locks the solver's inputs. This locks the data elements that have been set into the solver as well as prohibits new data elements being set through a call to `SetInputData`.

```cpp
HRESULT LockInputs();
```

**Return values**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
IRSolverInputs

Remarks

This is usually implemented with a lock count. LockInputs increments the count, and UnlockInputs decrements the count. The inputs are locked if the count is non-zero.

1.38.5 IRSolverInputs::SetInputData

Assigns a data element to an input to the solver.

HRESULT SetInputData(
    [in] UINT ulnput,
    [in] LPUNKNOWN pSource,
    [in] UINT uElementNum
);

Parameters

ulnput

[in] Index of the input on the solver to which the data element will be assigned. This value must be less than the number returned from GetInputCount.

pSource

[in] Pointer to the data source that manages the data element to assign to the input.

uElementNum

[in] Index of the data element within the data source specified by the pSource parameter. The solver calls IRDataSource::GetDataElementAccessor with this index to retrieve the data element.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The index parameter was invalid.</td>
</tr>
<tr>
<td>RSOLVE_E_OUTOFORDER</td>
<td>The input was set out of a required order for the solver.</td>
</tr>
<tr>
<td>RSOLVE_E_SOLVING</td>
<td>The solver was solving, and could not accept new inputs.</td>
</tr>
<tr>
<td>RSOLVE_E_INPUTSLOCKED</td>
<td>The inputs were locked, and could not be set.</td>
</tr>
<tr>
<td>RSOLVE_E_WRONGNUMBEROFDIMENSIONS</td>
<td>The data element contained the wrong number of dimensions for the desired input.</td>
</tr>
</tbody>
</table>
IRSolverMapping

<table>
<thead>
<tr>
<th>RSOLVE_E_WRONGDIMENSIONSIZE</th>
<th>The data element contained a dimension with the wrong size, relative to another input in the solver that uses a equivalent dimension.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSOLVE_E_WRONGDATATYPE</td>
<td>The solver could not convert from the type of data in the data element to the type required by the solver for this input.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.38.6 IRSolverInputs::UnlockInputs

Unlocks the solver's inputs. This unlocks the data elements that have been set into the solver as well as enables new data elements being set through a call to SetInputData.

HRESULT UnlockInputs();

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This is usually implemented with a lock count. LockInputs increments the count, and UnlockInputs decrements the count. The inputs are locked if the count is non-zero.

1.39 IRSolverMapping

Interface IRSolverMapping provides the basic functionality for all mappings in the framework. Clients can specify the data element that a mapping will use to map data into a solver, as well as specify how the data should be maintained.

When to implement. Implement this interface on any mapping object within the framework.

When to use. Clients use this interface to specify the data element that the mapping will map, and to specify the preference for where that data should be maintained. Inbound solver sites use this interface to manage the mapping activity itself, and to lock and unlock the mapping.

Methods in vtable order

IUnknown methods:
- QueryInterface
- AddRef

Description
- Returns pointers to supported interfaces.
- Increments reference count.
IRSolverMapping

IRSolverMapping methods:
- LockMapping
- UnlockMapping
- MappingLocked
- MapData
- GetDataSource
- SetDataSource
- GetSolverSiteIn
- SetSolverSiteIn
- GetMaintainPreference
- SetMaintainPreference

1.39.1 IRSolverMapping::GetDataSource

Retrieves the data source and the index of a data element within that data source that is managed by
the mapping.

HRESULT GetDataSource{
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid,
    [out] UINT* puElement
};

Parameters

riid
[in] Identifier of the interface to return on the data source that manages the mapping’s data element.

ppVoid
[out] Pointer to a location that upon successful completion contains the requested interface pointer
to the data source that manages the mapping’s data element. upon failure, *ppVoid is set to zero.

puElement
[out] Pointer to a UINT that upon successful completion contains the index of the data element
within the data source that corresponds to the mapping’s data element. Upon failure, *puElement is
set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOINTERFACE</td>
<td>The data source does not support the requested interface.</td>
</tr>
</tbody>
</table>
Remarks

A data element is contained within a data source, and is identified within the data source by its index. Each mapping is associated with a single data element, so in order to provide complete information for accessing a mapping’s data element, this method returns both the data source and the index within that data source that identifies the data element.

1.39.2 IRSolverMapping::GetMaintainPreference

Retrieves the client’s preference as to where the mapping should store the data element that is being set in the solver’s inputs when it maps data. The options are for the data source to maintain it, which requires that the data source can maintain it (see IRDataSource::CanMaintainSolverData), for the mapping to maintain it by creating a clone, and for the solver to maintain it internally.

    HRESULT GetMaintainPreference(  
        [out] DWORD* pPreference  
    );

Parameter

pPreference

[out] Pointer to a called-allocated DWORD that upon successful completion contains the client’s preference, as chosen from the RSOLVE_MAPPING_PREFERENCES enumeration.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.39.3 IRSolverMapping::GetSolverSiteIn

Returns the inbound solver site that contains this mapping.
IRSolverMapping

HRESULT GetSolverSiteIn(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);

Parameters

riid

[in] Identifier of the interface to return on the inbound solver site.

ppVoid

[out] Pointer to location that upon successful completion points to the requested interface on the inbound solver site. If the inbound solver site does not support the interface specified in iid, *ppVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_NOSOLVERSITE</td>
<td>The inbound solver site has not yet been set in the mapping.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The inbound solver site does not support the requested interface.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.39.4 IRSolverMapping::LockMapping

Locks the mapping. The mapping locks the data source on which it depends for data, so that the mapping will not receive any new data availability notifications.

HRESULT LockMapping();

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
Remarks

This is usually implemented with a lock count. LockMapping increments the count, and UnlockMapping decrements the count. The mapping is locked if the count is non-zero.

1.39.5 IRSolverMapping::MapData

Maps the data element from the data source managed by the mapping into the solver’s inputs, as determined by the mechanisms of the mapping object. This method is usually called only by the inbound solver site.

HRESULT MapData();

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_CANNOTGETINPUTDATA</td>
<td>The data element was not assigned to this mapping.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

This method may also return any of the errors from IRSolver::SetInputData.

Remarks

This method usually calls IRSolver::SetInputData to assign the data element into the solver.

1.39.6 IRSolverMapping::MappingLocked

Returns whether the mapping is currently locked or not.

HRESULT MappingLocked(
    [out] DWORD* pfLocked
);

Parameter

pfLocked

[out] Pointer to a DWORD that upon successful completion is zero if the mapping is not locked or non-zero if the mapping is locked. This value is set to zero if the operation fails.
IRSolverMapping

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The mapping maintains a (non-negative) lock count, which is incremented by calls to IRSolverMapping::LockMapping and decremented by calls to IRSolverMapping::UnlockMapping. The mapping is locked if the lock count is non-zero.

1.39.7 IRSolverMapping::SetDataSource

Assigns a data source and a data element managed by the data source to the mapping.

```c++
HRESULT SetDataSource(
    [in] LPUNKNOWN pUnk,
    [in] UINT uElementNum
);
```

Parameters

- `pUnk` [in] Pointer to the data source.
- `uElementNum` [in] Index of the data element within the collection of data elements managed by the data source. This index identifies the data element.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The data element index was invalid for the provided data source.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The data source pointer was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
Remarks

The mapping can recover the data element from the data source by calling IRDataSource::GetDataElementAccessor and passing it the index number.

1.39.8 IRSolverMapping::SetMaintainPreference

Sets the client’s preference as to where the mapping should store the data element that is being set in the solver’s inputs when it maps data. The options are for the data source to maintain it, which requires that the data source can maintain it (see IRDataSource::CanMaintainSolverData), for the mapping to maintain it by creating a clone, and for the solver to maintain it internally.

```
HRESULT SetMaintainPreference(
    [in] DWORD Preference
);
```

Parameter

Preference

[in] A DWORD value selected from the RSOLVE_MAPPING_PREFERENCES enumeration that specifies the client’s preference.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The Preference parameter was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.39.9 IRSolverMapping::SetSolverSiteIn

Assigns the inbound solver site that contains this mapping.

```
HRESULT SetSolverSiteIn(
    [in] LPUNKNOWN pUnk
);
```

Parameter

pUnk

[in] Pointer to the IUnknown implementation of the inbound solver site. This value may be zero, to indicate that the mapping should release its current inbound solver site if it has one.
IRSolverMappingMechanism

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The operation failed because the parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The mapping releases its current inbound solver site, if it has one, before assigning the new one.

1.39.10 IRSolverMapping::UnlockMapping

Unlocks the mapping. The mapping will unlock the data source upon which it depends for its data element, so that it can receive new data change notifications.

```
HRESULT UnlockMapping();
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This is usually implemented with a lock count. LockMapping increments the count, and UnlockMapping decrements the count. The mapping is locked if the count is non-zero.

1.40 IRSolverMappingMechanism

Interface IRSolverMapperMechanism is the simplest of mapping mechanism interfaces. It enables a client to specify to which input on a solver a mapping should map its data element in the IRSolverMapping::MapData method. This is the most common mapping mechanism. However, it is but one way that the mapping mechanism could be assigned.

**When to implement.** Implement this interface to provide one possible mapping mechanism on a mapping object when the mapping will map directly into a single solver input.

**When to use.** Use this interface to establish the mapping mechanism for mappings that map into a single solver input.

**Methods in vtable order**

118
**IRSolverMappingMechanism**

**IUnknown methods:**
- QueryInterface: Returns pointers to supported interfaces.
- AddRef: Increments reference count.
- Release: Decrements reference count.

**IRSolverMappingMechanism methods:**
- GetSolverInputNum: Returns solver input index to map into.
- SetSolverInputNum: Sets solver input index to map into.

### 1.40.1 IRSolverMappingMechanism::GetSolverInputNum

Returns the input index to which the mapping will map its data element.

```cpp
HRESULT GetSolverInputNum(
    [out] UINT* puNum
);```

**Parameter**

`puNum`

[out] Pointer to a UINT that upon successful completion contains the solver input index to which the mapping will assign its data element. Upon failure, `*puNum` is set to zero.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
<tr>
<td>RSOLVE_E_INPUTNOTSET</td>
<td>No input number has been assigned to this mapping.</td>
</tr>
</tbody>
</table>

### 1.40.2 IRSolverMappingMechanism::SetSolverInputNum

Assigns an input index to the mapping. The mapping will assign its data element to the specified input in the implementation of IRSolverMapping::MapData.

```cpp
HRESULT SetSolverInputNum(
    [in] UINT uNum
);```

**Parameter**

`uNum`
IRSolverOutputInfo

[in] The input index to assign to the mapping mechanism. This value must be between zero and one less than the number of inputs to the solver.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The input index was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.41 IRSolverOutputInfo

Interface IRSolverOutputInfo provides information about an output provided by the solver. It is implemented within the context of SolverInfo objects for a solver.

When to implement. Implement this interface on sub-objects of the primary SolverInfo object that provide information about outputs of a solver. It is necessary to implement this interface on these sub-objects for solvers that provide their own implementation of the SolverInfo protocol. The generic SolverInfo implementation provides an implementation of this interface, and for solvers that use the generic SolverInfo implementation, it is not necessary to implement this interface.

When to use. Clients use this interface to discover properties of an output generated by a solver. In particular, this interface is useful for displaying these properties to the user through a solver browser.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
<tr>
<td>IRSolverBaseInfo methods:</td>
<td></td>
</tr>
<tr>
<td>GetName</td>
<td>Returns the name of the object.</td>
</tr>
<tr>
<td>GetShortDescription</td>
<td>Returns a short description of the object.</td>
</tr>
<tr>
<td>GetFlags</td>
<td>Returns flags about the object.</td>
</tr>
<tr>
<td>GetHelpFile</td>
<td>Returns name of help file describing object.</td>
</tr>
<tr>
<td>GetHelpContext</td>
<td>Returns help context within help file that describes object.</td>
</tr>
<tr>
<td>IRSolverOutputInfo methods:</td>
<td></td>
</tr>
<tr>
<td>GetDimInfoCount</td>
<td>Returns number of dimensions of output.</td>
</tr>
<tr>
<td>GetDimInfo</td>
<td>Returns dimension information object.</td>
</tr>
<tr>
<td>GetRetrievalFlags</td>
<td>Returns flags relating to retrieval capabilities.</td>
</tr>
<tr>
<td>GetChangeFlags</td>
<td>Returns flags relating to change capabilities.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>Returns data type of output data element.</td>
</tr>
</tbody>
</table>

1.41.1 IRSolverOutputInfo::GetChangeFlags

Returns flags describing the modification capabilities of an output of a solver.
HRESULT GetChangeFlags(
    [out] DWORD* pFlags
);  

Parameter

pFlags

[out] Pointer to a DWORD that upon successful completion contains the change flags. These values are from the enumeration RSOLVE_SOLVER_OUTPUT_CHANGE_FLAGS. Upon failure, *pFlags is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.41.2 IRSolverOutputlnfo::GetDataType

Returns the data type of the output.

HRESULT GetDataType(
    [out] LPRSOLVER_DATATYPE pType
);  

Parameter

pType

[out] Pointer to a RSOLVER_DATATYPE structure that upon successful completion contains the data type of the output. Upon failure, *pType is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The RSOLVER_DATATYPE typedefs to VARTYPE, the type field for VARIANT structures.
IRSolverOutputInfo

1.41.3 IRSolverOutputInfo::GetDimInfo

Returns the dimension information for a specified dimension of the output.

HRESULT GetDimInfo(
    [in] UINT index,
    [out] LPRSOLVERDIMINFO* ppDimInfo
);

Parameters

index

[in] Index of the dimension to retrieve the dimension information for. This value must be between zero and one less than the value returned from IRSolverOutputInfo::GetDimInfoCount.

ppDimInfo

[out] Pointer to a location which upon successful completion contains the IRSolverDimInfo pointer to the requested dimension information object. Upon failure, *ppDimInfo is set to zero.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The dimension index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.41.4 IRSolverOutputInfo::GetDimInfoCount

Returns the number of dimensions the data element has for this output.

HRESULT GetDimInfoCount(
    [out] UINT* puDims
);

Parameter

puDims

[out] Pointer to a UINT that upon successful completion contains the number of dimensions the data element has for this output. Upon failure, *puDims is set to zero.
Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

For scalar values, this should set *puDims to zero, not return an error.

1.41.5 IRSolverOutputInfo::GetRetrievalFlags

Returns flags describing the access capabilities of an output of a solver.

    HRESULT GetRetrievalFlags(
        [out] DWORD* pFlags
    );

Parameter

pFlags

[out] Pointer to a DWORD that upon successful completion contains the retrieval flags. Currently, there are no assigned retrieval flags, so servers should set *pFlags to zero. Upon failure, *pFlags is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.42 IRSolverOutputs

IRSolverOutputs is the primary solver interface for managing and accessing the outputs of a solver, implemented by all solver objects in the framework.

When to implement. Implement IRSolverOutputs on all solver objects and outbound solver sites. Outbound solver sites use IRSolverOutputs to wrap the output side of a solver.

When to use. Use IRSolverOutputs to access the outputs of a solver, including retrieving, locking, and unlocking the outputs.
IRSolverOutputs

Methods in vtable order

IUnknown methods:
- QueryInterface: Returns pointers to supported interfaces.
- AddRef: Increments reference count.
- Release: Decrements reference count.

IRSolverOutputs methods:
- LockOutputs: Locks the outputs.
- UnlockOutputs: Unlocks the outputs.
- OutputsLocked: Returns whether outputs are locked.
- GetOutputCount: Returns number of outputs.
- GetOutputData: Returns output data element.

1.42.1 IRSolverOutputs::GetOutputCount

Returns the number of outputs provided by this solver.

HRESULT GetOutputCount(
    [out] UINT* puOutputs
);

Parameter

puOutputs

[out] Pointer to a UINT that upon successful completion contains the number of outputs provided by
the solver. Upon failure, *puOutputs is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

If the solver has no outputs, *puOutputs is set to zero.

1.42.2 IRSolverOutputs::GetOutputData

Retrieves the data source and the index of a data element within that data source that corresponds to
an output on the solver.

HRESULT GetOutputData(
    [in] UINT uOutput,
);
IRSolverOutputs

[ in ] REFIID riid,
[ out, iid_is(riid)] LPVOID* ppVoid,
[ out] UINT* puElement
);

Parameters

uOutput

[in] The index of the output to retrieve the data source and data element index for. This value must be between zero and one less than the value returned from IRSolverOutputs::GetOutputCount().

riid

[in] Identifier of the interface to return on the data source that manages the desired output data element.

ppVoid

[out] Pointer to a location that upon successful completion contains the requested interface pointer to the data source that manages the desired output data element.

puElement

[out] Pointer to a UINT that upon successful completion contains the index of the data element within the data source that corresponds to the desired output.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The index uOutput is invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The data source does not support the requested interface.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>One of the output parameters was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

A data element is contained within a data source, and is identified within the data source by its index. Each solver output is associated with a single data element, so in order to provide complete information for accessing an output data element, this method returns both the data source and the index within that data source that identifies the data element.
1.42.3 IRSolverOutputs::LockOutputs

Locks the solver’s outputs. This locks the output data elements. A solver will typically call IRSolverInputs::LockInputs in the implementation of this method, so that while the outputs are locked, the inputs cannot be changed, ensuring data integrity.

```cpp
HRESULT LockOutputs();
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

This is usually implemented with a lock count. LockOutputs increments the count, and UnlockOutputs decrements the count. The outputs are locked if the count is non-zero.

1.42.4 IRSolverOutputs::OutputsLocked

Returns whether the outputs are currently locked or not.

```cpp
HRESULT OutputsLocked(
    [out] DWORD* pfLocked
);
```

**Parameter**

`pfLocked`

[out] Pointer to a DWORD that upon successful completion is zero if the outputs are not locked and non-zero if the outputs are locked. This value is set to zero if the operation fails.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

The solver maintains a (non-negative) lock count on its outputs, which is incremented by calls to IRSolverOutputs::LockOutputs and decremented by calls to IRSolverOutputs::UnlockOutputs. The outputs are locked if the lock count is non-zero.
1.42.5 IRSolverOutputs::UnlockOutputs

Unlocks the solver’s outputs. This unlocks the output data elements. A solver will typically call IRSolverInputs::UnlockInputs in the implementation of this method, so that once the outputs are unlocked, the inputs can be changed.

```c
HRESULT UnlockOutputs();
```

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

This is usually implemented with a lock count. LockOutputs increments the count, and UnlockOutputs decrements the count. The outputs are locked if the count is non-zero.

1.43 IRSolverParameters

Interface IRSolverParameters provides the capability to set and retrieve the parameters that configure a solver.

**When to implement.** Implement this interface on all solver objects in the framework.

**When to use.** Clients use this interface to configure a solver prior to execution. Sometimes, parameters may also provide output information, such as statistics of execution like run-time or optimality gaps.

**Methods in vtable order**

1. **IUnknown methods:**
   - QueryInterface
   - AddRef
   - Release

2. **IRSolverParameters methods:**
   - GetParameterCount
   - GetParameter
   - SetParameter

**Description**

- Returns pointers to supported interfaces.
- Increments reference count.
- Decrements reference count.
- Returns number of parameters.
- Returns a parameter.
- Sets a parameter.

1.43.1 IRSolverParameters::GetParameter

Returns the value of a parameter provided by the solver.

```c
HRESULT GetParameter(
    [in] UINT ulindex,
```
IRSolverParameters

    [out, retval] VARIANT* pVar

Parameters

ulndex

[in] A UINT specifying the index of the parameter to get. This value ranges from zero to one less than the value returned from GetParameterCount.

pVar

[out] Pointer to a caller-allocated VARIANT structure that upon successful completion contains the value of the requested parameter.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The parameter index was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.43.2 IRSolverParameters::GetParameterCount

Returns the number of parameters provided by the solver.

    HRESULT GetParameterCount(
        [out, retval] UINT* puCount
    );

Parameter

puCount

[out] Pointer to a UINT that upon successful completion contains the number of parameters provided by the solver. Upon failure, *puCount is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
</tbody>
</table>
Remarks

If the solver has no parameters, *puCount is set to zero.

1.43.3 IRSolverParameters::SetParameter

Sets the value of a parameter to configure the solver.

```
HRESULT SetParameter(
    [in] UINT ulIndex,
    [in] VARIANT Parameter
);
```

Parameters

ulIndex

[in] A UINT specifying the index of the parameter to set. This value ranges from zero to one less than the value returned from GetParameterCount.

Parameter

[in] A VARIANT containing the new value of the parameter.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The parameter was successfully assigned.</td>
</tr>
<tr>
<td>RSOLVE_E_INVALIDINDEX</td>
<td>The parameter index was invalid.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The parameter value is invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

The operation may also return any of the errors from the COM function VariantChangeType.

1.44 IRSolverParametersInterface

Interface IRSolverParametersInterface enables a client to access the parameters of a solver using a dispatch interface, rather than using the IRSolverParameters interface. Through the dispatch interface returned from IRSolverParametersInterface's single method, GetParametersInterface, the client can
IRSolverParamInfo

call IDispatch::Invoke using one of the parameter numbers and get or set parameters as if they were properties.

**When to implement.** Implement this interface on a solver object if the solver provides an IDispatch implementation of the parameters for that solver.

**When to use.** Clients should use this interface if they wish to acquire the IDispatch implementation of the parameters for a solver instead of using the IRSolverParameters interface.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

**IRSolverParametersInterface method:**

| GetParametersInterface | Returns dispatch implementation of parameters. |

1.44.1 **IRSolverParametersInterface::GetParametersInterface**

Returns the dispatch interface that implements the dispatch version of the parameters of the solver.

```c
HRESULT GetParametersInterface(
    [out] LPDISPATCH* ppDisp
);
```

**Parameter**

**ppDisp**

[out] Pointer to location that upon successful completion contains the dispatch interface pointer of the object that implements the solver’s parameters.

**Return values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.45 **IRSolverParamInfo**

Interface IRSolverParamInfo provides information about a parameter within the solver. It is implemented within the context of SolverInfo objects for a solver.
**When to implement.** Implement this interface on sub-objects of the primary SolverInfo object that provide information about parameters of a solver. It is necessary to implement this interface on these sub-objects for solvers that provide their own implementation of the SolverInfo protocol. The generic SolverInfo implementation provides an implementation of this interface, and for solvers that use the generic SolverInfo implementation, it is not necessary to implement this interface.

**When to use.** Clients use this interface to discover properties of a parameter that can configure a solver. In particular, this interface is useful for displaying these properties to the user through a solver browser.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRSolverBaselnfo methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetName</td>
<td>Returns the name of the object.</td>
</tr>
<tr>
<td>GetShortDescription</td>
<td>Returns a short description of the object.</td>
</tr>
<tr>
<td>GetFlags</td>
<td>Returns flags about the object.</td>
</tr>
<tr>
<td>GetHelpFile</td>
<td>Returns name of help file describing object.</td>
</tr>
<tr>
<td>GetHelpContext</td>
<td>Returns help context within help file that describes object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRSolverParamInfo methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDataType</td>
<td>Returns data type of parameter.</td>
</tr>
<tr>
<td>GetDefaultValueAsString</td>
<td>Returns default value.</td>
</tr>
<tr>
<td>GetTagStrings</td>
<td>Returns string/cookie array of possible parameter values.</td>
</tr>
<tr>
<td>GetTagValue</td>
<td>Returns parameter value given cookie from GetTagStrings.</td>
</tr>
</tbody>
</table>

### 1.45.1 IRSolverParamInfo::GetDataType

Returns the data type of the parameter.

```
HRESULT GetDataType(
    [out] VARTYPE pType
);
```

**Parameter**

* pType

[out] Pointer to a VARTYPE structure that upon successful completion contains the data type of the parameter.

**Return values**

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
</tbody>
</table>
IRSolverParamlnfo

| E_FAIL | The operation failed due to an unspecified reason. |

1.45.2 IRSolverParamlnfo::GetDefaultValueAsString

Returns the default value of the parameter in a human-readable string.

```
HRESULT GetDefaultValueAsString(
    [out, string] LPOLESTR* ppwszValue
);
```

**Parameter**

`ppwszValue`

[out] Points to a location that upon successful completion points to a zero-terminated string containing the string of the default value. Upon failure, *ppwszValue is set to zero.

**Return values**

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

This method allocates memory for the string returned in the `ppwszValue` parameter using the IMalloc::Alloc method. The caller is responsible for calling the IMalloc::Free method to free the string. Both the caller and this method use the OLE task allocator provided by a call to CoGetMalloc.

1.45.3 IRSolverParamlnfo::GetTagStrings

Returns a counted array of string pointers (LPOLESTR pointers). The strings pointed to provide a list of names that each correspond to values that the parameter can accept.

```
HRESULT GetTagStrings(
    [out] CALPOLESTR * pcaStringsOut,
    [out] CADWORD * pcaCookiesOut
);
```
Parameters

pcaStringsOut

[out] Pointer to a caller-allocated, counted array structure that contains the element count and address of a method-allocated array of string pointers. This method also allocates memory for the string values containing the predefined names, and it stores the string pointers in the array. If the method fails, no memory is allocated, and the contents of the structure are undefined.

pcaCookiesOut

[out] Pointer to the caller-allocated, counted array structure that contains the element count and address of a method-allocated array of DWORDs. The DWORD values in the array can be passed to IRSolverParamInfo::GetTagValue to retrieve the value associated with the name in the same array position inside pcaStringsOut. If the method fails, no memory is allocated, and the contents of the structure are undefined.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The arrays were allocated and filled successfully.</td>
</tr>
<tr>
<td>E_NOTIMPL</td>
<td>The parameter does not support predefined string names.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>One of the output parameters was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The structures CALPOLESTR and CADWORD are standard COM structures; see the COM reference for their definition. This method has the same semantics as the COM method IPerPropertyBrowsing::GetPreDefinedStrings.

Each string returned in the array pointed to by pcaStringsOut has a matching token in the counted array pointed to by pcaCookiesOut, where the token can be passed to IRSolverParamInfo::GetTagValue to get the actual value (an RSOLVER_DATA) corresponding to the string.

Using the predefined strings, a caller can obtain a list of strings for populating user interface elements, such as a drop-down listbox. When the end user selects one of these strings as a value to assign to a property, the caller can then obtain the corresponding value through IRSolverParamInfo::GetTagValue.
Notes to callers. Both the CALPOLESTR and CADWORD structures passed to this method are caller-allocated. The caller is responsible for freeing each string pointed to from the CALPOLESTR array as well as the CALPOLESTR structure.

All memory is allocated with CoTaskMemAlloc. The caller is responsible for freeing the strings and the array with CoTaskMemFree.

Upon return from this method, the caller is responsible for all this memory and must free it when it is no longer needed. Code to achieve this appears as follows:

```c
CALPOLESTR castr;
CADWORD cadw;
ULONG i;

pIRSolverParamInfo->GetTagStrings(&castr, &cadw);

// Use the strings and the cookies here...
CoTaskMemFree((void *)cadw.pElems);
for (i=0; i < castr.cElems; i++)
    CoTaskMemFree((void *)castr.pElems[i]);
CoTaskMemFree((void *)castr.pElems);
```

Notes to implementers. Support for predefined names and values is not required. If the parameter does not support these names, return E_NOTIMPL from this method. If this method is not implemented, IRSolverParamInfo::GetTagValue must not be implemented either.

This method fills the cElems and pElems fields of the CADWORD and CALPOLESTR structures. It allocates the arrays pointed to by these structures with CoTaskMemAlloc and fills those arrays. In the CALPOLESTR case, this method also allocates each string with CoTaskMemAlloc, storing each string pointer in the array.

1.45.4 IRSolverParamInfo::GetTagValue

Returns a RSOLVER_DATA structure (VARIANT) containing the value of the parameter. This property is associated with a predefined string name as returned from IRSolverParamInfo::GetTagStrings. The predefined string is identified by a token returned from GetTagStrings.

```c
HRESULT GetTagValue(
    [in] DWORD dwCookie,
    [out] VARIANT* pValOut
);
```

Parameters

dwCookie
Token identifying which value to return. The token was previously returned in the pcaCookiesOut array filled by IRSolverParamInfo::GetTagStrings.

pValOut

Pointer to a VARIANT that upon successful completion contains the value for the parameter that corresponds to the cookie.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The cookie was not valid for this parameter.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
<tr>
<td>E_NOTIMPL</td>
<td>The parameter does not support predefined string names.</td>
</tr>
</tbody>
</table>

Remarks

Notes to callers. The caller is responsible for freeing any allocations contained in the VARIANT. Unless the vt field of VARIANT is VT_VARIANT, the caller can free memory using a single call to VariantClear. Otherwise, the caller must recursively free the contained VARIANTs before freeing the outer VARIANT.

Notes to implementers. Support for predefined names and values is not required. If your object does not support these names, return E_NOTIMPL from this method. If this method is not implemented, IRSolverParamInfo::GetTagStrings must not be implemented either.

This method allocates any memory needed inside the VARIANT structure.

1.46 IRSolverProvideInfo

IRSolverProvideInfo enables a client to access the SolverInfo introspection object from a solver that the client already has a pointer to. Solvers provide this interface if they implement their own, custom per-solver SolverInfo object, or if they wish to provide optimized access to the generic SolverInfo implementation.

When to implement. Implement this interface on a solver if the solver implements its own, per-solver SolverInfo objects, or to provide optimized access for a client to the generic SolverInfo implementation.
IRSolverProvideInfo

When to use. Clients should use this interface if they already have an interface pointer to a solver, in order to access the solver’s SolverInfo. Clients should use this interface prior to using the generic SolverInfo library functions, as the solver may have its own technique for acquiring the SolverInfo object.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRSolverProvideInfo method:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSolverInfo</td>
<td>Returns the solver’s SolverInfo object.</td>
</tr>
</tbody>
</table>

1.46.1 IRSolverProvideInfo::GetSolverInfo

Returns the SolverInfo object that describes the solver.

HRESULT GetSolverInfo(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);

Parameters

riid

[in] Identifier of the interface to return from the SolverInfo object. Typically this is either IID_IRSolverInfo or IID_IRSolverDescription.

ppVoid

[out] Pointer to location that upon successful completion contains the requested interface pointer of the SolverInfo object.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The SolverInfo does not support the requested interface.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
Remarks

The SolverInfo and SolverDescription interfaces can both be obtained from this function by changing the requested interface identifier.

1.47 IRSolverRegistration

Interface IRSolverRegistration enables a client or solver to register and unregister a solver on the system. This interface is part of the core services.

When to implement. Generally, it is not necessary to implement this interface. The core services implementation of the SolverRegistrar implements this interface.

When to use. Use this interface to register or unregister a solver on the system. The core services implementation of this interface can be acquired by creating the SolverRegistrar object, which has CLSID CLSID_RSolverRegistrar.

Methods in vtable order

Unknown methods:
- QueryInterface
- AddRef
- Release

IRSolverRegistration methods:
- RegisterSolver
- RegisterSolverWithOptions
- UnregisterSolver

Description
- Returns pointers to supported interfaces.
- Increments reference count.
- Decrements reference count.
- Registers a solver with the system.
- Registers a solver and its SolverInfo with the system.
- Unregisters a solver with the system.

1.47.1 IRSolverRegistration::RegisterSolver

Registers a solver into the global directory of installed solvers.

HRESULT RegisterSolver(
    [in] REFCLSID rclsidSolver
);

Parameter

rclsidSolver

[in] The CLSID of the solver to register.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>
IRSolverRegistration

<table>
<thead>
<tr>
<th>E_INVALIDARG</th>
<th>The solver's CLSID was invalid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.47.2 IRSolverRegistration::RegisterSolverWithInfo

Registers a solver and its associated generic SolverInfo implementation into the global directory of installed solvers.

```c
HRESULT RegisterSolverWithInfo(
    [in] REFCLSID rclsidSolver,
    [in] REFGUID rguidSolverInfo
);
```

Parameters

- `rclsidSolver`
  
  [in] The CLSID of the solver to register.

- `rguidSolverInfo`
  
  [in] The GUID of the SolverInfo Type Library that describes the SolverInfo and optionally the SolverDescription of the solver.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>One of the inputs was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.47.3 IRSolverRegistration::UnregisterSolver

Removes a solver from the global directory of installed solvers.

```c
HRESULT UnregisterSolver(
    [in] REFCLSID rclsidSolver
);
```

Parameter

- `rclsidSolver`
The CLSID of the solver to remove from the directory of installed solvers.

Return values

<table>
<thead>
<tr>
<th>CLSID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>REGDB_E_CLASSNOTREG</td>
<td>The solver was not registered in the directory to begin with.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.48 IRSolverSetInfo

Interface IRSolverSetInfo provides information about a set used by the solver. It is implemented within the context of SolverInfo objects for a solver.

When to implement. Implement this interface on sub-objects of the primary SolverInfo object that provide information about sets of a solver. It is necessary to implement this interface on these sub-objects for solvers that provide their own implementation of the SolverInfo protocol. The generic SolverInfo implementation provides an implementation of this interface, and for solvers that use the generic SolverInfo implementation, it is not necessary to implement this interface.

When to use. Clients use this interface to discover properties of a set used by a solver. In particular, this interface is useful for displaying these properties to the user through a solver browser.

Methods in vtable order

IUnknown methods:
- QueryInterface: Returns pointers to supported interfaces.
- AddRef: Increments reference count.
- Release: Decrements reference count.

IRSolverBaseInfo methods:
- GetName: Returns the name of the object.
- GetShortDescription: Returns a short description of the object.
- GetFlags: Returns flags about the object.
- GetHelpFile: Returns name of help file describing object.
- GetHelpContext: Returns help context within help file that describes object.

IRSolverSetInfo method:
- GetSetFlags: Returns flags relating to set details.

1.48.1 IRSolverSetInfo::GetSetFlags

Returns flags describing the nature and capabilities of a set for a solver.

HRESULT GetSetFlags(
    [out] DWORD* pFlags
);

139
Parameter

pFlags

[out] Pointer to a DWORD that upon successful completion contains the set flags. Currently, there are no assigned set flags, so servers should set *pFlags to zero. Upon failure, *pFlags is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.49 IRSolverSiteIn

IRSolverSiteIn is the primary inbound solver site interface. All inbound solver sites, or objects acting as inbound solver sites, must implement this interface. IRSolverSiteIn provides access to the solver and outbound solver site associated with this inbound solver site.

Note that inbound solver sites also have the responsibility to implement a number of other interfaces, including IRSolverInputs and IRSolverSiteInMappings, in order to completely wrap the solvers they contain. An inbound solver site is also responsible for initiating solver execution.

**When to implement.** Implement this interface on any inbound solver site implementation, as well as on any object that acts as its own inbound solver site.

**When to use.** A client typically uses this interface while creating, wiring, and destroying a solution network. Any object can use this interface to access the solver or outbound solver site associated with this inbound solver site.

**Methods in vtable order**

```plaintext
IUnknown methods:                        Description
   QueryInterface                        Returns pointers to supported interfaces.
   AddRef                                Increments reference count.
   Release                               Decrements reference count.

IRSolverSiteIn methods:                  Description
   GetSolverSiteOut                      Returns the corresponding outbound solver site.
   SetSolverSiteOut                      Sets the corresponding outbound solver site.
   GetSolver                             Returns the solver wrapped by this solver site.
   SetSolver                             Sets the solver wrapped by this solver site.
   IsSolverInputAvailable                Returns whether an input is available for the solver.
```

1.49.1 IRSolverSiteIn::GetSolver

Returns the solver wrapped by this inbound solver site.
HRESULT GetSolver(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);

Parameters

riid

[in] Identifier of the interface to request on the solver.

ppVoid

[out] Pointer to location that upon successful completion points to the requested interface. If the object does not support the interface specified in iid, *ppvVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>S_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSOLVE_E_NOSOLVER</td>
<td>The solver has not yet been set in the solver site or created by it.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The solver does not support the requested interface.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.49.2 IRSolverSiteIn::GetSolverSiteOut

Returns the outbound solver site associated with this inbound solver site.

HRESULT GetSolverSiteOut(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);

Parameters

riid

[in] Identifier of the interface to request on the outbound solver site.

ppVoid
IRSolverSiteIn

[out] Pointer to location that upon successful completion points to the requested interface. If the outbound solver site does not support the interface specified in iid, *ppvVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_NOSOLVERSITE</td>
<td>The outbound solver site has not yet been set in the inbound solver site.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The outbound solver site does not support the requested interface.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.49.3 IRSolverSiteIn::IsSolverInputAvailable

Returns whether a specific input for the solver is available to be mapped.

```
HRESULT IsSolverInputAvailable(
    [in] UINT ulnputIndex,
    [out] DWORD* pfAvailable
);
```

Parameters

ulnputIndex

[in] The index to check for availability. This value must be less than the number of inputs specified by IRSolverInputs::GetInputCount.

pfAvailable

[out] Pointer to a DWORD that upon successful completion is non-zero if the input is available to be mapped or zero if the input is not available. Upon failure, *pfAvailable is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

142
1.49.4 IRSolverSiteIn::SetSolver

Assigns a solver to the inbound solver site.

```c
HRESULT SetSolver(
    [in] LPUNKNOWN pUnk
);
```

**Parameter**

**pUnk**

[in] Pointer to the IUnknown implementation of the solver. This value may be zero, to indicate that the solver site should release its current solver if it has one.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The operation failed because the parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

The solver site releases its current solver, if it has one, before assigning the new one. If the solver object is valid, this function should call IRSolver::SetSolverSiteIn passing itself as the inbound solver site.

1.49.5 IRSolverSiteIn::SetSolverSiteOut

Assigns an outbound solver site to be associated with this inbound solver site.

```c
HRESULT SetSolverSiteOut(
    [in] LPUNKNOWN pUnk
);
```

**Parameter**

**pUnk**

[in] Pointer to the IUnknown implementation of the outbound solver site. This value may be zero, to indicate that the inbound solver site should release its current outbound solver site if it has one.
**IRSolverSiteInMappings**

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The operation failed because the parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

**Remarks**

The inbound solver site releases its current outbound solver site, if it has one, before assigning the new one.

### 1.50 IRSolverSiteInMappings

The interface IRSolverSiteInMappings enables a client to manage a set of mappings contained at an inbound solver site. A client can add, remove, and enumerate over the mappings currently at a solver site.

**When to implement.** Implement this interface on an inbound solver site object to provide access to the mappings contained at the solver site.

**When to use.** A client uses this interface in the course of creating and wiring up a network and in destroying the network. Other objects that need to access the mappings at an inbound solver site can use this interface to acquire the number of contained mappings and a pointer to each one.

**Methods in vtable order**

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

**IRSolverSiteInMappings methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddMapping</td>
<td>Adds a mapping to the solver site.</td>
</tr>
<tr>
<td>RemoveMapping</td>
<td>Removes a mapping from the solver site.</td>
</tr>
<tr>
<td>GetMappingCount</td>
<td>Returns number of mappings in the solver site.</td>
</tr>
<tr>
<td>GetMapping</td>
<td>Returns a mapping by its cookie.</td>
</tr>
<tr>
<td>GetMappingByIndex</td>
<td>Returns a mapping by its index.</td>
</tr>
<tr>
<td>RemoveAllMappings</td>
<td>Removes all mappings from the solver site.</td>
</tr>
</tbody>
</table>

### 1.50.1 IRSolverSiteInMappings::AddMapping

Adds a mapping to the inbound solver site's container of mappings.

```c
HRESULT AddMapping(
    [in] LPUNKNOWN pUnk,
    [out] DWORD* pdwCookie
)```

144
Parameters

pUnk

[in] Pointer to the mapping to add to the inbound solver site’s collection.

pdwCookie

[out] Pointer to a DWORD that upon successful completion contains a cookie identifying this mapping within the collection. Future access to the mapping through the solver site relies on this cookie value. Upon failure, *pdwCookie is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The mapping parameter was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The client should store the cookie returned to it, as it is required for future access to the mapping within the solver site.

1.50.2 IRSolverSiteInMappings::GetMapping

Given a cookie assigned in IRSolverSiteInMappings::AddMapping, retrieves the mapping designated by the cookie.

```cpp
HRESULT GetMapping(
    [in] DWORD dwCookie,
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);
```

Parameters

_dwCookie

[in] A DWORD cookie previously obtained when adding the mapping via a call to IRSolverSiteInMappings::AddMapping.
IRSolverSiteInMappings

riid

[in] Identifier of the interface to retrieve on the mapping.

ppVoid

[out] Pointer to a location that upon successful completion contains a pointer to the requested interface on the mapping. Upon failure, *ppVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The cookie was invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The mapping does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.50.3IRSolverSiteInMappings::GetMappingByIndex

Retrieves a mapping by its index within the array of mappings contained by the solver site. This function is useful primarily for enumerating the mappings.

HRESULT GetMappingByIndex(
    [in] UINT ulIndex,
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);

Parameters

ulIndex

[in] The index of the mapping to retrieve. This value must be between zero and one less than the value returned by IRSolverSiteInMappings::GetMappingCount.

riid

[in] Identifier of the interface to retrieve from the mapping.

ppVoid
[out] Pointer to a location that upon successful completion contains a pointer to the requested interface on the mapping. Upon failure, *ppVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The cookie was invalid.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The mapping does not support the requested interface.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

While the inbound solver site may store the mappings in any order, it will always iterate through the mappings in the same order. That is, if mapping \( x < mapping_y \), then it will always be so, even as mappings are removed or added between them.

1.50.4 IRSolverSiteInMappings::GetMappingCount

Returns the number of mappings currently contained in the inbound solver site.

```cpp
HRESULT GetMappingCount(
    [out] UINT* puMappings
);
```

Parameter

puMappings

[out] Pointer to a UINT that upon successful completion contains the number of mappings currently contained in the inbound solver site. Upon failure, *puMappings is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>


IRSolverSitelnSolverFactory

### 1.50.5 IRSolverSitelnMappings::RemoveAllMappings

Removes all mappings from the inbound solver site.

```cpp
HRESULT RemoveAllMappings();
```

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

### 1.50.6 IRSolverSitelnMappings::RemoveMapping

Removes a mapping from the inbound solver site.

```cpp
HRESULT RemoveMapping(
    [in] DWORD dwCookie
);
```

**Parameter**

- `dwCookie`
  - [in] A DWORD cookie that designates the mapping to remove, previously acquired via a call to IRSolverSitelnMappings::AddMapping.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The cookie parameter was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

### 1.51 IRSolverSiteInSolverFactory

Interface IRSolverSiteInSolverFactory allows a client to defer creation of the solver wrapped by an inbound solver site to the inbound solver site itself. Therefore, solvers can be created only upon execution, and so consume fewer resources prior to execution.

**When to implement.** Implement this interface an in inbound solver site that supports being a solver factory.
When to use. A client can use this interface to allow the inbound solver site to control the creation and destruction of the solver that it wraps.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRSolverSiteInSolverFactory methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpecifySolver</td>
<td>Specifies CLSID of the solver.</td>
</tr>
<tr>
<td>SpecifySolverProgID</td>
<td>Specifies ProgID of the solver.</td>
</tr>
<tr>
<td>CreateSolver</td>
<td>Creates the solver immediately.</td>
</tr>
<tr>
<td>DestroySolver</td>
<td>Releases the solver immediately.</td>
</tr>
</tbody>
</table>

1.51.1 IRSolverSiteInSolverFactory::CreateSolver

Instructs the inbound solver site to immediately create the solver, regardless of what the flag settings were in a prior call to IRSolverSiteInSolverFactory::SpecifySolver or IRSolverSiteInSolverFactory::SpecifySolverProgID.

HRESULT CreateSolver();

Return values

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.51.2 IRSolverSiteInSolverFactory::DestroySolver

Instructs the inbound solver site to release its reference to the solver, effectively, for the purposes of the solution network, destroying it.

HRESULT DestroySolver();

Return values

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
IRSolverSiteInSolverFactory

Remarks

Most reference-counted COM objects can destroy and deallocate only themselves. Hence, this method does not guarantee that the solver object will be destroyed, but rather that for the purposes of the solution network, the solver is no longer accessible.

Note to implementers. This method usually just calls IRSolverSiteIn::SetSolver(NULL) on itself and IRSolverSiteOut::SetSolver(NULL) on the outbound solver site associated with this inbound solver site.

1.51.3 IRSolverSiteInSolverFactory::SpecifySolver

Specifies the CLSID and execution context of the solver that the inbound solver site will create.

HRESULT SpecifySolver(
    [in] REFCLSID rclsid,
    [in] DWORD dwClsCtx,
    [in] DWORD dwFlags
);

Parameters

rclsid
[in] The CLSID of the solver that the inbound solver site will create.

dwClsCtx
[in] A DWORD containing values from the CLSCTX enumeration that specifies the execution context of the solver. See the COM function CoCreateInstanceEx for more details.

dwFlags
[in] A DWORD containing flags providing instructions on when the inbound solver site should create the solver. These flags can be taken from the enumeration RSOLVE_SOLVERFACTORY_CREATEFLAGS.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>One of the parameters was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
Remarks

Calling this method does not necessarily cause the immediate creation of the solver, although it might. The inbound solver site does not need to check at this time whether the CLSID is valid and whether the solver can be created.

1.51.4 IRSolverSiteInSolverFactory::SpecifySolverProgID

Specifies the ProgID and execution context of the solver that the inbound solver site will create.

    HRESULT SpecifySolverProgID(
        [in, string] LPOLESTR pwszProgId,
        [in] DWORD dwClsCtx,
        [in] DWORD dwFlags
    );

Parameters

pwszProgId

[in] Pointer to a zero-terminated string containing the ProgID of the solver that the inbound solver site will create.

dwClsCtx

[in] A DWORD containing values from the CLSCTX enumeration that specifies the execution context of the solver. See the COM function CoCreateInstanceEx for more details.

dwFlags

[in] A DWORD containing flags providing instructions on when the inbound solver site should create the solver. These flags can be taken from the enumeration RSOLVE_SOLVERFACTORY_CREATEFLAGS.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>One of the parameters was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

This operation may also return any of the error codes from the COM function StringFromCLSID.
Remarks

Calling this method does not necessarily cause the immediate creation of the solver, although it might. The inbound solver site does not need to check at this time whether the CLSID is valid and whether the solver can be created.

1.52 IRSolverSiteOut

Interface IRSolverSiteOut is the primary outbound solver site interface. All outbound solver sites, or objects acting as outbound solver sites, must implement this interface. IRSolverSiteOut provides access to the solver and inbound solver site associated with this outbound solver site.

Note that outbound solver sites also have the responsibility to implement a number of other interfaces, including IRSolverOutputs and IRDataSource, in order to completely wrap the solvers they contain. An outbound solver site is also responsible for receiving the solver notifications through the interface IRSolverAdvise.

When to implement. Implement this interface on any outbound solver site implementation, as well as on any object that acts as its own outbound solver site.

When to use. A client typically uses this interface while creating, wiring, and destroying a solution network. Any object can use this interface to access the solver or inbound solver site associated with this outbound solver site.

Methods in vtable order

<table>
<thead>
<tr>
<th>IUnknown methods:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
</tbody>
</table>

IRSolverSiteOut methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSolverSiteIn</td>
<td>Returns corresponding inbound solver site.</td>
</tr>
<tr>
<td>SetSolverSiteIn</td>
<td>Sets corresponding inbound solver site.</td>
</tr>
<tr>
<td>GetSolver</td>
<td>Returns solver wrapped by the solver site.</td>
</tr>
<tr>
<td>SetSolver</td>
<td>Sets solver wrapped by the solver site.</td>
</tr>
</tbody>
</table>

1.52.1 IRSolverSiteOut::GetSolver

Returns the solver wrapped by this outbound solver site.

HRESULT GetSolver(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
)

Parameters

riid
[in] Identifier of the interface to request on the solver.

ppvVoid

[out] Pointer to location that upon successful completion points to the requested interface. If the object does not support the interface specified in iid, *ppvVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_NOSOLVER</td>
<td>The solver has not yet been set in the solver site.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The solver does not support the requested interface.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

1.52.2 IRSolverSiteOut::GetSolverSiteIn

Returns the inbound solver site associated with this outbound solver site.

HRESULT GetSolverSiteIn(
    [in] REFIID riid,
    [out, iid_is(riid)] LPVOID* ppVoid
);

Parameters

riid

[in] Identifier of the interface to request on the outbound solver site.

ppvVoid

[out] Pointer to location that upon successful completion points to the requested interface. If the inbound solver site does not support the interface specified in iid, *ppvVoid is set to zero.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>RSOLVE_E_NOSOLVERSITE</td>
<td>The inbound solver site has not yet been set in the outbound solver site.</td>
</tr>
</tbody>
</table>
1.52.3 IRSolverSiteOut::SetSolver

Assigns a solver to the outbound solver site.

HRESULT SetSolver(
    [in] LPUNKNOWN pUnk
);

Parameter

pUnk

[in] Pointer to the IUnknown implementation of the solver. This value may be zero, to indicate that the solver site should release its current solver if it has one.

Return values

<table>
<thead>
<tr>
<th>E_OK</th>
<th>The operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_INVALIDARG</td>
<td>The operation failed because the parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The solver site releases its current solver, if it has one, before assigning the new one. If the solver object is valid, this function should call IRSolver::SetSolverSiteOut passing itself as the outbound solver site.

1.52.4 IRSolverSiteOut::SetSolverSiteIn

Assigns an inbound solver site to be associated with this outbound solver site.

HRESULT SetSolverSiteIn(
    [in] LPUNKNOWN pUnk
);
Parameter

pUnk

[in] Pointer to the IUnknown implementation of the inbound solver site. This value may be zero, to indicate that the outbound solver site should release its current inbound solver site if it has one.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>The operation failed because the parameter was invalid.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

Remarks

The outbound solver site releases its current inbound solver site, if it has one, before assigning the new one.

1.53 IRSolverStatus

The interface IRSolverStatus enables an object to asynchronously query the status of a solver at any time during its execution, or before or after execution. Solvers that can be queried asynchronously typically register themselves in the running object table (ROT). Clients can then acquire pointers to the solvers through the ROT, and then query them through the IRSolverStatus interface. This is currently the only interface through which a solver can reliably be accessed asynchronously by any object.

When to implement. Implement this interface on a solver if it supports asynchronous status queries.

When to use. This interface is typically used by clients that have no prior association with a solver, but need to connect to them at random times to check on their status. Another object in the same solution network could also use this interface to asynchronously query the solver at any time.

Note to implementers. Usually, implementing this interface requires separate threads for solver processing and for asynchronous solver queries and notifications.

Methods in vtable order

<table>
<thead>
<tr>
<th>method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IUnknown methods: QueryInterface</td>
<td>Returns pointers to supported interfaces.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments reference count.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements reference count.</td>
</tr>
<tr>
<td>IRSolverStatus method: GetStatus</td>
<td>Returns the solver’s status.</td>
</tr>
</tbody>
</table>
IRSolverStatus

1.53.1 IRSolverStatus::GetStatus

Returns the current status, asynchronously, of the solver at the time of the query.

```cpp
HRESULT GetStatus(
    [out] DWORD* pdfStatus
);
```

Parameter

`pdfStatus`

[out] Pointer to a DWORD that upon successful completion contains the status of the solver. The status can be one of the values in the RSOLVE_SOLVER_STATUS enumeration. Upon failure, *pdfStatus is set to zero.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
2 Global Library Functions

This section describes the global functions provided by the core services.

2.1 RLoadRegSolverInfo

Loads the SolverInfo object, using the generic implementation, for the specified SolverInfo GUID, version number, and locale.

```
RSOLVEAPI RLoadRegSolverInfo(
    REFGUID rguid,
    unsigned short wVerMajor,
    unsigned short wVerMinor,
    LCID lcid,
    IRSolverInfo** ppInfo
);
```

Parameters

rguid

[in] The GUID of the SolverInfo object to load. This is the type library identifier of the SolverInfo Type Library that defines the SolverInfo for the desired solver.

wVerMajor

[in] Major version number of the SolverInfo Type Library to load.

wVerMinor

[in] Minor version number of the SolverInfo Type Library to load.

lcid

[in] Locale identifier of the SolverInfo Type Library to load.

ppinfo

[out] Pointer to a location that upon successful completion contains the IRSolverInfo interface pointer for the requested SolverInfo object.

Return values

| S_OK        | The operation completed successfully. |
RLoadSolverInfo

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

This function may also return any of the error codes from the COM function LoadRegTypeLib.

Remarks

This function is similar to the COM function LoadRegTypeLib, and uses it in its implementation to load the underlying type library that describes the SolverInfo object.

2.2 RLoadSolverInfo

Given the file name of a SolverInfo Type Library, loads the SolverInfo object described by that type library.

```c
RSOLVEAPI RLoadSolverInfo(
    LPOLESTR pwszFile,
    IRSolverlnfo** pplnfo
);
```

Parameters

pwszFile

[in] Pointer to a zero-terminated string that names the type library file.

pplnfo

[out] Pointer to a location that upon successful completion contains the IRSolverInfo interface pointer for the requested SolverInfo object.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>

This function may also return any of the error codes from the COM function LoadTypeLib.
Remarks

This function is similar to the COM function \texttt{LoadTypeLib}, and calls \texttt{LoadTypeLib} to load the underlying type library that describes the \texttt{SolverInfo} object.

2.3 \texttt{RLoadSolverInfoClsid}

Given the CLSID of a solver, loads the \texttt{SolverInfo} object registered with that solver. The solver must have registered itself using \texttt{IRSolverRegistration::RegisterSolverWithInfo} or set up the proper registry entries itself during installation for this function to work. This function is a simple wrapper for a call to \texttt{RLookupInfoGuid} followed by a call to \texttt{RLoadRegSolverInfo}.

\begin{verbatim}
RSOLVEAPI RLoadSolverInfoClsid(
    REFCLSID rclsidSolver,
    LCID Icid,
    IRSolverInfo ** pplinfo
);
\end{verbatim}

Parameters

\begin{description}
\item[rclsidSolver] [in] The CLSID of the solver whose associated SolverInfo will be loaded.
\item[Icid] [in] A locale identifier specifying which SolverInfo Type Library should be loaded.
\item[pplinfo] [out] Pointer to a location that upon successful completion contains the \texttt{IRSolverInfo} interface pointer for the requested SolverInfo object.
\end{description}

Return values

\begin{center}
\begin{tabular}{|c|l|}
\hline
\textbf{S\_OK} & The operation completed successfully. \\
\hline
\texttt{REGDB\_E\_CLASSNOTREG} & The solver’s CLSID was invalid. \\
\hline
\texttt{E\_OUTOFMEMORY} & The operation failed due to insufficient memory. \\
\hline
\texttt{E\_POINTER} & The output parameter was an invalid pointer. \\
\hline
\texttt{E\_FAIL} & The operation failed due to an unspecified reason. \\
\hline
\end{tabular}
\end{center}
RLookupInfoGuid

2.4 RLookupInfoGuid

Given the CLSID of a solver, looks up the GUID and version information of the SolverInfo registered with the solver. The solver must have registered itself using IRSolverRegistration::RegisterSolverWithInfo or set up the proper registry entries itself during installation for this function to work.

RSOLVEAPI RLookupInfoGuid(
    REFCLSID rclsidSolver,
    GUID* pGuidInfo,
    unsigned short* pwVerMajor,
    unsigned short* pwVerMinor
);

Parameters

rclsidSolver

[in] The CLSID of the solver whose associated SolverInfo will be loaded.

pGuidInfo

[ out] Pointer to a caller-allocated GUID that upon successful completion contains the GUID of the SolverInfo Type Library registered with the solver.

pwVerMajor

[ out] Pointer to a caller-allocated WORD that upon successful completion contains the major version number of the SolverInfo Type Library registered with the solver.

pwVerMinor

[ out] Pointer to a caller-allocated WORD that upon successful completion contains the minor version number of the SolverInfo Type Library registered with the solver.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>REGDB_E_CLASSNOTREG</td>
<td>The solver CLSID was invalid.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>The output parameter was an invalid pointer.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>The operation failed due to an unspecified reason.</td>
</tr>
</tbody>
</table>
3 Structures

This section describes structures defined by the framework.

3.1 RDESCPROPIDINFO

Describes a description property registered with the core services. See IEnumRDESCPROPIDINFO and IRDescriptionPropEnumeration::EnumRDESCPROPIDINFO.

typedef struct tagRDESCPROPIDInfo {
    GUID rdescpropid;
    LCID lcid;
    OLECHAR pwszDescription[108];
} RDESCPROPIDINFO, *LPRDESCPROPIDINFO;

Members

rdescpropid

Description property identifier of the description property being described.

lcid

Locale identifier of the description property.

pwszDescription

Human-readable string providing the name of a description of the description property.

3.2 RDESCRIPTIONPROPERTY

Describes a description property exposed by a solver. See IEnumRDESCRIPTIONPROPERTY and IRSolverDescriptionProperties::EnumDescriptionProperties.

typedef struct tagRDESCRIPTIONPROPERTY {
    RDESCPROPID rdpidPropId;
    VARIANT varProp;
    OLECHAR pwszPropName[128];
} RDESCRIPTIONPROPERTY, *LPRDESCRIPTIONPROPERTY;

Members

rdpidPropld

161
The description property identifier of this description property.

varProp
The value of the description property.

pwszPropName
A human-readable name of the description property identifier. This is so that the client does not have to look up the name using the description property manager provided by the core services.

### 3.3 RREGSOLVERINFO

Provides the CLSID of a solver and the GUID of its SolverInfo. See IEnumRREGSOLVERINFO and IRSolverEnumeration::EnumSolversWithInfo.

```c
typedef struct tagRREGSOLVERINFO {
    CLSID clsidSolver;
    GUID guidSolverInfo;
} RREGSOLVERINFO, *LPRREGSOLVERINFO;
```

#### Members

- **clsidSolver**
  
  CLSID of a solver installed on the system.

- **guidSolverInfo**
  
  GUID of the SolverInfo associated with the specified solver.
4 Enumerations

This section describes the various enumerations used in the framework.

4.1 RSD_DESCRIPTIONFLAGS

Used by IRSolverDescription::GetSolutionFlags, this enumeration provides flags used by the SolverDescription object.

Values

RSD_GENERATES_FEASIBLE_SOLUTIONS (0x00000001)
The solver generates solutions that are feasible.

RSD_GENERATES_INFEASIBLE_SOLUTIONS (0x00000002)
The solver generates solutions that are infeasible. The combination of this flag with the previous indicates that the solutions generated might be feasible or infeasible.

RSD_GENERATES_OPTIMAL_SOLUTIONS (0x00000004)
The solver generates solutions that are optimal.

RSD_GENERATES_NONOPTIMAL_SOLUTIONS (0x00000008)
The solver generates solutions that are non-optimal. The combination of this flag with the previous indicates that the solutions generated might be optimal or non-optimal.

4.2 RSI_SOLVERFLAGS

Used by IRSolverInfo::GetFlags, this enumeration provides flags used by a solver SolverInfo object.

Values

RSIASYCHRONOUS_INPUTS (0x00010000)
The solver accepts inputs asynchronously. That is, a client or mapping may assign inputs to the solver at any time.

RSI_ORDERED_INPUTS (0x00000200)
The solver requires that the inputs be assigned in their index order.

RSIASYCHRONOUS_EXECUTION (0x00000400)
RSOLVE_DATA_ACCESS

The solver supports asynchronous execution through the IRSolverAsynchSolve interface.

RSL_SYNCHRONOUS_EXECUTION (0x00000800)

The solver supports synchronous execution through the IRSolver::Solve method.

4.3 RSOLVE_DATA_ACCESS

Used by IRDataElementAccess::GetSubElementAt, IRDataElement2::GetColumn, and IRDataElement2::GetRow, this enumeration provides flags that specify how sub-element access occurs.

Values

RDA_COPY (0x00000000)

The sub-element access should create a copy of the sub-element. Changes to the sub-element object will not change the original data element.

RDA_CONTAINED (0x00000001)

The sub-element access should reference the contained sub-element. Changes to the sub-element object will change the original data element.

4.4 RSOLVE_MAPPING_PREFERENCES

Used by IRSolverMapping::GetMaintainPreference and IRSolverMapping::SetMaintainPreference, this enumeration provides flags used to specify where a mapping should maintain the data for an input to a solver.

Values

RMP_UNKNOWN (0x00000000)

The client has no preference.

RMP_DATASOURCE (0x00000001)

The client prefers that the data source maintains the data.

RMP_MAPPING (0x00000002)

The client prefers that the mapping clones the data and maintains it.

RMP_SOLVER (0x00000003)

The client prefers that the solver maintains the data.
4.5 RSOLVE_SOLVE_COMPLETE_RESULTS

Used by IRSolverAdvise::OnSolveComplete, this enumeration provides completion flags used by a solver.

Values

- **RSCR_OK** (0x00000000)
  The solver completed successfully.

- **RSCR_OPTIMIZED** (0x00000001)
  The solver found an optimal solution. The interpretation of “optimal” depends on the solver.

- **RSCR_CANCELLED** (0x00000002)
  The solver processing was cancelled.

- **RSCR_OUTOFTIME** (0x00000003)
  The solver exceeded its time limit without generating a solution.

- **RSCR_MAXITERATIONS** (0x00000004)
  The solver exceeded its iteration limit without generating a solution.

- **RSCR_APPROXIMATED** (0x00000005)
  The solver found an approximate solution. The interpretation of “approximate” depends on the solver.

4.6 RSOLVE_SOLVE_NOTIFY_ACTIONS

Used by IRSolverAdvise::OnSolveNotify, this enumeration provides flags used by all solvers during progress updates to specify an action the solver should take after sending a notification.

Values

- **RSNA_CONTINUE** (0x00000000)
  The solver should continue executing.

- **RSNA_CANCEL** (0x00000001)
  The solver should cancel its execution.

- **RSNA_STOPNOTIFY** (0x80000000)
RSOLVE_SOLVE_NOTIFY_FLAGS

Used in conjunction with either RSNA_CONTINUE or RSNA_CANCEL, instructs the solver (or whoever is managing the update mechanism) to cancel the update, and not send any further notifications to other advise sinks.

4.7 RSOLVE_SOLVE_NOTIFY_FLAGS

Used by IRSolverAdvise::OnSolveNotify, this enumeration provides flags used by all solvers during progress updates.

Values

RSNF_PERCENTAGE (0x00000001)
The dPercComplete parameter in the call to IRSolverAdvise::OnSolveNotify is valid.

RSNF_HOURS (0x00000002)
The IHoursLeft parameter in the call to IRSolverAdvise::OnSolveNotify is valid.

RSNF_MINUTES (0x00000004)
The IMinutesLeft parameter in the call to IRSolverAdvise::OnSolveNotify is valid.

RSNF_MILLISECONDS (0x00000008)
The IMillisecondsLeft parameter in the call to IRSolverAdvise::OnSolveNotify is valid.

RSNF_TIME (0x0000000E)
All of the time parameters in the call to IRSolverAdvise::OnSolveNotify are valid.

4.8 RSOLVE_SOLVER_ASSIGNMENT_FLAGS

Used by IRSolverInputInfo::GetAssignmentFlags, this enumeration provides assignment flags for a solver input.

Values

RSAF_CANUNLOCKAFTERSETTING (0x00000100)
The solver either copies the data or locks it itself upon assignment, so the client, or whoever calls IRSolver::SetInputData, may unlock the data element after assigning it to the solver.

RSAF_OPTIONAL (0x00000200)
The input to the solver is optional; if IRSolver::SetInputData is not called for this input, the solver will use a default data element.
4.9 RSOLVE_SOLVER_INFO_FLAGS

Used by IRSolverBaseInfo::GetFlags, this enumeration provides flags used by all SolverInfo objects.

Values

RSI_ENUMERATED  (0x00000001)
The item is an enumerated object.

RSI_EXPERT  (0x00000002)
The item should be displayed in SolverInfo browsers only in “expert” modes. In non-expert modes, the item should be hidden.

RSI_HIDDEN  (0x00000004)
The item should be hidden in SolverInfo browsers.

4.10 RSOLVE_SOLVER_INPUT_CHANGE_FLAGS

Used by IRSolverInputInfo::GetChangeFlags, this enumeration provides the change flags for an input to the solver.

Values

RSICF_READONLY  (0x00001000)
The input is read only; that is, once the data element has been set through a call to IRSolver::SetInputData, the data element will be locked and cannot be changed.

RSICF_CANGET  (0x00002000)
The input, once assigned, is accessible from the solver. Calls to IRSolverInputs::GetInputData will succeed. If this flag is not set, calls to IRSolverInputs::GetInputData will fail.

4.11 RSOLVE_SOLVER_OUTPUT_CHANGE_FLAGS

Used by IRSolverOutputInfo::GetChangeFlags, this enumeration provides the change flags for an output to the solver.

Value

RSOCF_READONLY  (0x00001000)
The output is read only; that is, the data element will be locked and cannot be changed.
RSOLVE_SOLVER_STATUS

4.12 RSOLVE_SOLVER_STATUS

Used by IRSolverStatus::GetStatus, this enumeration provides flags that provide the status of a solver.

Values

RSS_UNINITIALIZED [0x00000001]
The solver is in an uninitialized state, and might have received some but not all of its inputs.

RSS_INPUTSVALID [0x00000002]
The inputs to the solver are all valid.

RSS_SOLVING [0x00000004]
The solver is currently executing.

RSS_PAUSED [0x00000008]
The solver is currently paused.

RSS_OUTPUTSVALID [0x00000010]
The outputs of the solver are all valid.

RSS_DESTROYING [0x00000020]
The solver is currently being destroyed.

4.13 RSOLVE_SOLVERFACTORY_CREATEFLAGS

Used by IRSolverSiteInSolverFactory::SpecifySolver and IRSolverSiteInSolverFactory::SpecifySolverProgID, this enumeration provides flags that specify whether and when the inbound solver site should create the solver it wraps.

Values

RSFC_NORMAL [0x00000000]
The inbound solver site should not create the solver, and instead should rely on the client setting the solver with a call to IRSolverSiteIn::SetSolver.

RSFC_IMMEDIATE [0x00010000]
The inbound solver site should create the solver during the call to IRSolverSiteInSolverFactory::SpecifySolver or IRSolverSiteInSolverFactory::SpecifySolverProgID.

RSFC_LAZY [0x00020000]
The inbound solver site should create the solver whenever it has the opportunity.

**RSFC_LASTPOSSIBLE**

(0x00040000)

The inbound solver site should delay the creation of the solver as long as possible.

### 4.14 RSOLVE_SOLVERFACTORY_DESTROYFLAGS

Used by `IRSolverSiteInSolverFactory::SpecifySolver` and `IRSolverSiteInSolverFactory::SpecifySolverProglD`, this enumeration provides flags that specify whether and when the inbound solver site should destroy the solver it wraps.

#### Values

**RSFD_NORMAL**

(0x00000000)

The inbound solver site should not destroy the solver *per se*, but should rely on the client to clear the solver with a call to `IRSolverSiteIn::SetSolver`.

**RSFD_ONSOLVECOMPLETE**

(0x00000001)

The inbound solver site should destroy the solver after receiving the `IRSolverAdvise::OnSolveComplete` notification.

**RSFD_ONSITEDESTROY**

(0x00000002)

The inbound solver site should destroy the solver when it is itself destroyed.
5 Constants

This section lists the constants defined by the framework.

5.1 Interface identifiers (IIDs)

These section lists the interface identifiers of the interfaces defined by the framework.

IID_IEnumRDESCPROPIDINFO .......... 08E99001-F674-11D1-91ED-00207810C741
IID_IEnumRDESCRIPTIONPROPERTY ....... 08E99002-F674-11D1-91ED-00207810C741
IID_IEnumRREGSOLVERINFO ............ 08E99003-F674-11D1-91ED-00207810C741
IID_IRDataAdvise ...................... 08E99004-F674-11D1-91ED-00207810C741
IID_IRDataAdviseHolder ............... 08E99005-F674-11D1-91ED-00207810C741
IID_IRDataElement ..................... 08E99006-F674-11D1-91ED-00207810C741
IID_IRDataElementl .................... 08E99007-F674-11D1-91ED-00207810C741
IID_IRDataElement2 .................... 08E99008-F674-11D1-91ED-00207810C741
IID_IRDataElementAccess .............. 08E99009-F674-11D1-91ED-00207810C741
IID_IRDataElementClone .......... 08E9900A-F674-11D1-91ED-00207810C741
IID_IRDataElementCreator .......... 08E9900B-F674-11D1-91ED-00207810C741
IID_IRDataElementItemProperties ..... 08E9900C-F674-11D1-91ED-00207810C741
IID_IRDataElementItemPropertiesChange 08E9900D-F674-11D1-91ED-00207810C741
IID_IRDataElementScalar ............ 08E9900E-F674-11D1-91ED-00207810C741
IID_IRDataSource ...................... 08E9900F-F674-11D1-91ED-00207810C741
IID_IRDescriptionPropEnumeration ... 08E99010-F674-11D1-91ED-00207810C741
IID_IRDescriptionPropRegistration ... 08E99011-F674-11D1-91ED-00207810C741
IID_IRDimension ....................... 08E99012-F674-11D1-91ED-00207810C741
IID_IRDimensionClone ................. 08E99013-F674-11D1-91ED-00207810C741
IID_IRDimensionCreator .............. 08E99014-F674-11D1-91ED-00207810C741
IID_IRMappingAdvise .................. 08E99015-F674-11D1-91ED-00207810C741
IID_IRSet .................. 08E99016-F674-11D1-91ED-00207810C741
IID_IRSetClone ....................... 08E99017-F674-11D1-91ED-00207810C741
IID_IRSetCreator ..................... 08E99018-F674-11D1-91ED-00207810C741
IID_IRSetModifier .................... 08E99019-F674-11D1-91ED-00207810C741
IID_IRSolver ......................... 08E9901A-F674-11D1-91ED-00207810C741
IID_IRSolverAdvise .................... 08E9901B-F674-11D1-91ED-00207810C741
IID_IRSolverAdviseHolder ............ 08E9901C-F674-11D1-91ED-00207810C741
IID_IRSolverAsynchSolve ............. 08E9901D-F674-11D1-91ED-00207810C741
IID_IRSolverBaseInfo ................. 08E9901E-F674-11D1-91ED-00207810C741
IID_IRSolverControl ................. 08E9901F-F674-11D1-91ED-00207810C741
IID_IRSolverDescription .......... 08E99020-F674-11D1-91ED-00207810C741
IID_IRSolverDescriptionProperties .. 08E99021-F674-11D1-91ED-00207810C741
IID_IRSolverDimInfo ................. 08E99022-F674-11D1-91ED-00207810C741
IID_IRSolverDimInfoDisp .......... 08E99023-F674-11D1-91ED-00207810C741

170
5.2 Class identifiers (CLSIDs)

This section lists the CLSIDs of the main objects implemented by the core services and the default networking objects, like the solver sites and mappings.

<table>
<thead>
<tr>
<th>CLSID (Name)</th>
<th>CLSID Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLSID_RDataAdviseHolder</td>
<td>08E99101-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RDescriptionPropManager</td>
<td>08E99102-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RMapping</td>
<td>08E99103-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RSolverAdviseHolder</td>
<td>08E99104-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RSolverInfo</td>
<td>08E99105-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RSolverInfoDisp</td>
<td>08E99106-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RSolverRegistrar</td>
<td>08E99107-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RSolverSiteIn</td>
<td>08E99108-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RSolverSiteOut</td>
<td>08E99109-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RSolverSiteInMappings</td>
<td>08E9910A-F674-11D1-91ED-00207810C741</td>
</tr>
<tr>
<td>CLSID_RSolverSiteOut</td>
<td>08E9910B-F674-11D1-91ED-00207810C741</td>
</tr>
</tbody>
</table>
Other GUIDs

5.3 Other GUIDs

This section lists other GUIDs used by the framework, including the description property identifiers, LIBIDs of objects provided by the core services, and category identifiers.

<table>
<thead>
<tr>
<th>GUID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATID_RSolver</td>
<td>CatID_RSolver</td>
</tr>
<tr>
<td>GUID_RDEFAULTVALUE</td>
<td>GUID_RDEFAULTVALUE</td>
</tr>
<tr>
<td>GUID_RINPUTOPTIONAL</td>
<td>GUID_RINPUTOPTIONAL</td>
</tr>
<tr>
<td>GUID_RPOSSIBLEVALUES</td>
<td>GUID_RPOSSIBLEVALUES</td>
</tr>
<tr>
<td>GUID_RSAF_CANMAINTAINDATA</td>
<td>GUID_RSAF_CANMAINTAINDATA</td>
</tr>
<tr>
<td>GUID_RSAF_CANUNLOCKAFTERSETTING</td>
<td>GUID_RSAF_CANUNLOCKAFTERSETTING</td>
</tr>
<tr>
<td>GUID_RSAF_OPTIONAL</td>
<td>GUID_RSAF_OPTIONAL</td>
</tr>
<tr>
<td>GUID_RSI_ASYNCHRONOUS_EXECUTION</td>
<td>GUID_RSI_ASYNCHRONOUS_EXECUTION</td>
</tr>
<tr>
<td>GUID_RSI_ASYNCHRONOUS_INPUTS</td>
<td>GUID_RSI_ASYNCHRONOUS_INPUTS</td>
</tr>
<tr>
<td>GUID_RSI_EXPERT</td>
<td>GUID_RSI_EXPERT</td>
</tr>
<tr>
<td>GUID_RSI_HIDDEN</td>
<td>GUID_RSI_HIDDEN</td>
</tr>
<tr>
<td>GUID_RSI_ORDERED_INPUTS</td>
<td>GUID_RSI_ORDERED_INPUTS</td>
</tr>
<tr>
<td>GUID_RSI_SYNCHRONOUS_EXECUTION</td>
<td>GUID_RSI_SYNCHRONOUS_EXECUTION</td>
</tr>
<tr>
<td>GUID_RSICF_CANGET</td>
<td>GUID_RSICF_CANGET</td>
</tr>
<tr>
<td>GUID_RSICF_READONLY</td>
<td>GUID_RSICF_READONLY</td>
</tr>
<tr>
<td>GUID_RSOCF_READONLY</td>
<td>GUID_RSOCF_READONLY</td>
</tr>
<tr>
<td>GUID_RSOLVERCLSID</td>
<td>GUID_RSOLVERCLSID</td>
</tr>
<tr>
<td>GUID_RSOLVERIMPLEMENTEDINTERFACE</td>
<td>GUID_RSOLVERIMPLEMENTEDINTERFACE</td>
</tr>
<tr>
<td>GUID_RSOLVERINPUTS</td>
<td>GUID_RSOLVERINPUTS</td>
</tr>
<tr>
<td>GUID_RSOLVEROUTPUTS</td>
<td>GUID_RSOLVEROUTPUTS</td>
</tr>
<tr>
<td>GUID_RSOLVERPARAMETERS</td>
<td>GUID_RSOLVERPARAMETERS</td>
</tr>
<tr>
<td>GUID_RSOLVERTYPELIB</td>
<td>GUID_RSOLVERTYPELIB</td>
</tr>
<tr>
<td>LIBID_RSOLVERINFOLib</td>
<td>LIBID_RSOLVERINFOLib</td>
</tr>
<tr>
<td>LIBID_RSOLVERSITELib</td>
<td>LIBID_RSOLVERSITELib</td>
</tr>
<tr>
<td>RDESCPROPID_Author</td>
<td>RDESCPROPID_Author</td>
</tr>
<tr>
<td>RDESCPROPID_Citation</td>
<td>RDESCPROPID_Citation</td>
</tr>
<tr>
<td>RDESCPROPID_Developer</td>
<td>RDESCPROPID_Developer</td>
</tr>
<tr>
<td>RDESCPROPID_SolutionFlags</td>
<td>RDESCPROPID_SolutionFlags</td>
</tr>
<tr>
<td>RDESCPROPID_URL</td>
<td>RDESCPROPID_URL</td>
</tr>
<tr>
<td>RDESCPROPID_WorstRuntime</td>
<td>RDESCPROPID_WorstRuntime</td>
</tr>
<tr>
<td>RDESCPROPID_WorstRuntimeAsTeX</td>
<td>RDESCPROPID_WorstRuntimeAsTeX</td>
</tr>
</tbody>
</table>

5.4 Error codes (HRESULTs)

This section lists the custom HRESULTs used by the framework.

```c
#define RSOLVE_ERROR(_x) MAKE_HRESULT(SEVERITY_ERROR, FACILITY_ITF, _x)
```

- `RSOLVE_E_CANNOTGETINPUTDATA` (0x1000)
- `RSOLVE_E_DATAELEMENTLOCKED` (0x1001)
- `RSOLVE_E_INPUTSLOCKED` (0x1002)
- `RSOLVE_E_INVALIDCOLUMN` (0x1003)

172
Error codes (HRESULTs)

RSOLVE_E_INVALIDINDEX.............................. RSOLVE_ERROR(0x1004)
RSOLVE_E_INVALIDROW .................................. RSOLVE_ERROR(0x1005)
RSOLVE_E_NOSOLVER .................................. RSOLVE_ERROR(0x1006)
RSOLVE_E_NOSOLVERSITE ................................ RSOLVE_ERROR(0x1007)
RSOLVE_E_NOTALLINPUTS ................................ RSOLVE_ERROR(0x1008)
RSOLVE_E_NOTSUPPORTED ................................ RSOLVE_ERROR(0x1009)
RSOLVE_E_OUTPUTNOTSET ................................ RSOLVE_ERROR(0x100A)
RSOLVE_E_OUTPUTSLOCKED ................................ RSOLVE_ERROR(0x100B)
RSOLVE_E_READONLY .................................. RSOLVE_ERROR(0x100C)
RSOLVE_E_SETLOCKED .................................. RSOLVE_ERROR(0x100D)
RSOLVE_E_SOLVING .................................. RSOLVE_ERROR(0x100E)
RSOLVE_E_WRONGDATATYPE ................................ RSOLVE_ERROR(0x100F)
RSOLVE_E_WRONGDIMENSIONSIZE ..................... RSOLVE_ERROR(0x1010)
RSOLVE_E_WRONGNUMBEROFDIMENSIONS ............... RSOLVE_ERROR(0x1011)