SAP Data Provider

HELP.BCCIDATAPROV

Release 4.6C

Copyright

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Contents

SAP Data Provider

Purpose

The SAP Data Provider (referred to in this documentation as the Data Provider) is a standard data retrieval interface used by controls in the SAP GUI.

The Data Provider:

- Manages the data
- Maps the data on to standard Windows data types and data objects
- Controls data conversion

The Data Provider consists of:

- ABAP function modules

  ABAP function modules are the Data Provider's interface to the application programmer or shell programmer. They use normal ABAP data types such as internal tables.

- SAP GUI function calls

  The SAP GUI contains 2 RFC-enabled functions. One function sends a table to the SAP GUI, the other function gets a table from the SAP GUI. The table passed by this call is a data object.

- Local tables object

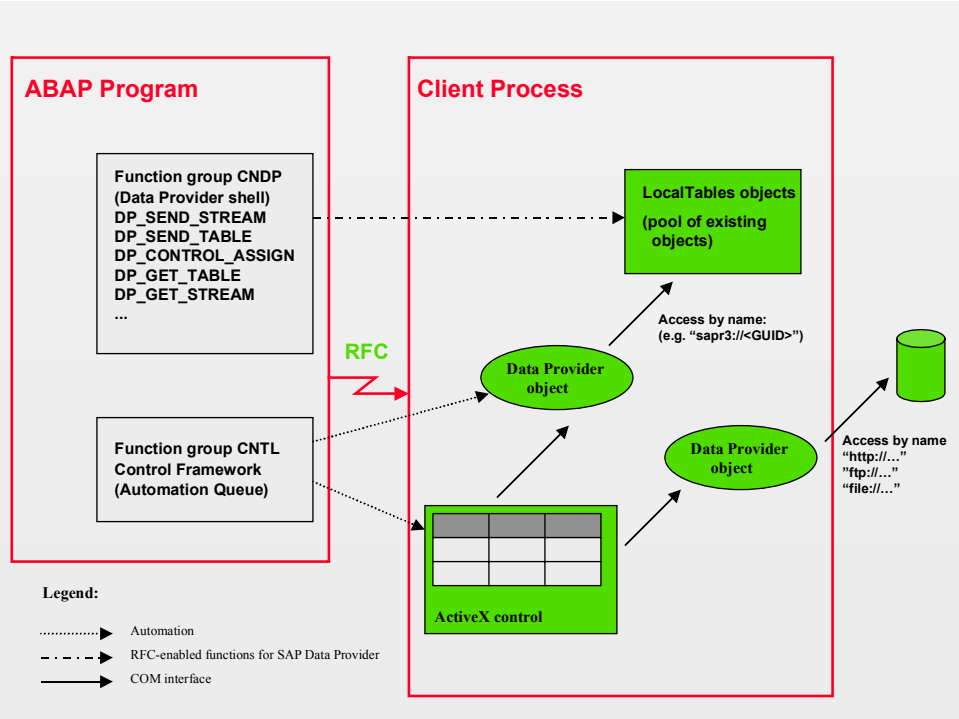
  The local tables object manages all tables in a data pool, which is global within a single process space.

- Data Provider object

  The Data Provider object provides access to the data objects in the local tables object. You can either reference data in the local tables object or store references to data there.

  The Data Provider object uses standard Windows data types and data objects, so users require no knowledge of internal tables or ABAP data types.

  The Data Provider object also provides access to ftp: and http: servers, as well as files stored on local servers or file servers.



The Data Provider is used by most ActiveX controls supplied by SAP. Examples of such controls are the Editor Control, HTML Control, Image Control, Tree Control, and Desktop OfficeIntegration.

Implementation Considerations

You need the Data Provider because:

- The standard automation queue mechanism does not allow you to pass amounts of data exceeding 256 bytes through method calls, nor is it able to pass Binary Large Object (BLOB) data such as office documents, screen shots and editor contents.
- External data sources (that is, sources that exist outside R/3 or the current ABAP program) must be accessible. Examples of these external data sources are the Info Repository or local files.

Constraints

The SAP Data Provider is available only on Windows platforms (Windows95 or Windows NT 4.0+).

Functional Description

The SAP Data Provider is based on the push model. This means that, before a control can reference a data object, the data is always passed from the server to the client as an internal table (by RFC).

Each data object has a unique name, which usually has the syntax

```
sapr3://<GUID>
```

and is known as the URL. To reference a data object, you specify its URL.

Data on the client side is often represented as internal tables, but there are also typical interfaces for mass data. These forms of representation are known as **media**. When accessing data, you specify the desired medium. The data is then represented accordingly, and converted if necessary.

Each data object on the client side is described by the attributes *Type* and *SubType*, which are closely related to the MIME types used in the Internet and are known as the **format**. Conversions between different media depend on the format of the data object, since different conversions are available for different formats. You can also convert data objects between different formats, depending on the source and target formats.

When transporting data from the client to the server, the same principle applies. You can place data in the Data Provider object using any medium you like. If there is no URL, you must assign one to the data object via the SAP Data Provider. You can then specify this URL in order to pass the data object as an internal table to the application server.

To enable access to non-R/3 data sources, the SAP Data Provider can also process URLs in the http:, ftp:, and file: namespaces. The procedure for these is identical to that described above, except that the data is not passed from the application server to the client via a push model, but loaded in a pull model from the relevant server.

Use of MIME Types

To describe Binary Large Object (BLOB) data, MIME types are used. This allows you to manage the conversion of different types of BLOB data, and permits access to the data objects themselves.

By specifying a MIME type or part of a MIME type before accessing a server, you can ensure that a data transfer is triggered only if data of the relevant type exists.

If data exists in several different formats, specifying MIME types also allows you to specify which format is to be used. The rules governing this are listed in the table below:

MIME type specification	Processing
Type and sub-type specified	Only this MIME type is processed.
Either type or sub-type specified	Only data corresponding to the specified part is processed. The other part is ignored or interpreted as a wildcard.
Neither type nor sub-type specified	All data is processed. This can result in unnecessary server accesses and consequently worse performance.

You must specify a MIME type for BLOB data, otherwise the data is not accepted and an error occurs.

- If no suitable standard MIME type exists for a data object, you can use any name:

  - The main type should always be called **application**
  - The sub-type should always begin with **x-** (for example, application/x-myspecialtype).
- If the MIME type is irrelevant, use **application/x-unknown**.
- If you want to transport your own non-standard MIME type from/to an http: server, do one of the following:
  - Use the standard MIME type **application/octet-stream**
  - Register your own MIME type on the http: server.

When transporting BLOB data as an internal table to the client, you should specify the size of the BLOB data. If you fail to do this, the assumed size is the number of lines multiplied by the width of the table.

Typed Tables

When transporting typed tables, you need to know the structure in the client.

You always use **application/x-r3table** as MIME type. Also passed in this case is a table of the type RFC_FIELDS, which describes the actual data table. For this type of data object, there is a conversion object, which meets the requirements of typed tables.

To transport typed tables, there are function modules in the function group CNDP.

- When transporting typed tables from the client to an ABAP program, you should first transport an empty table of the same type to the client. This way, the program/control in the client can fill the table, and you can transport it back to the server any time you like.
- You cannot transport typed internal tables from or to http:, ftp: or file: servers, because this would require a known and secure protocol for serializing internal tables. It makes more sense to switch to OLE DB and Active Data Object (ADO). The required format conversion object is registered under the MIME type **application/x-rowset**.
- There is no delta management in RFC between the SAP GUI and the application server. For this reason, you are not recommended to send tables to the client, modify them, and then request them again from the application server. You can try to store the modification in your own table, but this does not make sense for BLOB data, because modifying BLOB data in ABAP programs is not always reliable.

Data Transfer from ABAP Program to Client

1. The ABAP program calls CONTROL_CREATE, or another create module, to create a control to handle the BLOB data.
2. The ABAP program reads the BLOB data into an internal table.
3. The ABAP program calls a function module such as DP_SEND_STREAM, which provides parameters for the internal table that holds the data, the size of the BLOB data in bytes, and the MIME type of the BLOB data.

a) The called function module gets some system information and calls DP_CREATE_URL.

b) DP_CREATE_URL generates a unique name (sapr3://<GUID>) for the current data object.

c) DP_PUT_CLIENT_TABLE is called in the SAP GUI (SAPAWRFC.DLL) and receives the internal table, the size and the MIME type of the data, the URL, and the system information.

i) DP_PUT_CLIENT_TABLE attaches the internal table to an ISAPrfcITab interface.

ii) A reference to the ISAPrfcITab interface, the MIME type, the size of the BLOB data and other system information are stored in the global object LocalTables together with the URL and also an identifier for the current context.

iii) DP_CREATE_URL returns the name (URL), under which a reference to the internal table was stored in LocalTables.

4. The control generated in point 1 knows and generates a Data Provider object. If not, go to point 5.

a) The URL of the data object is communicated to the control by CONTROL_SET_PROPERTY or CONTROL_CALL_METHOD.

i) The control generates a Data Provider object.

ii) In the Data Provider object, the control calls (for example) the method SetDataFromUrl, using the URL as parameter.

The Data Provider object generates a reference to the global LocalTables object.

The URL is passed to the LocalTables object. This gives the Data Provider object a reference to the ISAPrfcITab interface.

b) The control reads the data using a property of the Data Provider object.

i) The Data Provider object returns the data according to the access medium required by the control. Access to data is not necessarily via the ISAPrfcITab interface, it can be via any standard COM interface (IStream, ILockBytes, IUnknown, VARIANT,…)[1]

c) Go to point 8.

5. The ABAP program calls the function module DP_CREATE, which generates a Data Provider object in the client.

6. The ABAP program calls the method SetDataFromUrl of the Data Provider object and passes the URL as essential parameter.

a) The Data Provider object generates a reference to the global LocalTables object.

b) The URL is passed to the LocalTables object. This gives the Data Provider object a reference to the ISAPrfcITab interface and all additional information.

7. The control generated in point 1 does not know the SAP Data Provider.

a) The ABAP program calls CONTROL_ASSIGN_PROPERTY2, which assigns a property of the Data Provider object to a property of the control.

[1] Siehe **Error! Reference source not found.**

Depending on the type of data, the MIME type and the property (medium) used, the data is passed to the control.

8. The transaction ends.
9. A new transaction starts.
   a) The SAP GUI recognizes that the last transaction has ended.
   b) The SAP GUI generates a reference to the global LocalTables object and informs the object that the last transaction has finished by passing the context identifier of the finished transaction.
      i) The LocalTables object deletes all references to tables that have the specified context identifier and releases these references (Garbage Collection)

Other Data Sources

If the data object does not originate from the ABAP program, but from an external source, you can use the same process, with the following exceptions:

- Omit points 2 and 3 for transfer to the client
- Get the reference to the data object via the URL.

  At present, the ftp:, http:, and file: namespaces are supported.

Data Transfer from Client to ABAP Program

1. The ABAP program calls CONTROL_CREATE, or a similar module, to create a control.
2. The control generated in point 1 knows the Data Provider object. Otherwise, go to point 3.
   a) The control generates a Data Provider object.
   b) The control gives the Data Provider object a reference to its data in a suitable medium.
   c) The control calls the method SaveDataToUrl in the Data Provider object. Either the control specifies the URL, or the Data Provider object generates a URL with the syntax „sapr3://<GUID>“.
      i) The Data Provider object generates a reference to the global object LocalTables.
      ii) Both the URL and a reference to the data object are passed to this global object.
      iii) Go to point 4.
3. The control generated in point 1 does not know the Data Provider object.
   a) The ABAP program generates a Data Provider object with DP_CREATE.
   b) The ABAP program generates an appropriate format object.
   c) The ABAP program calls CONTROL_ASSIGN_PROPERTY2, which assigns a property of the control to a property of the format object. This control property contains the data or a reference to the data.
   d) The ABAP program calls the method SaveDataToUrl in the Data Provider object.

e) Continue as under 2.c.

4. The data is transferred to the ABAP program.

   a) The ABAP program calls the function module DP_GET_STREAM, which contains an internal table as a parameter to store the data and the URL of the data object from 2.c.

   b) DP_GET_STREAM calls the function DP_GET_CLIENT_TABLE in the SAP GUI (SAPAWRFC.DLL) and passes the URL and the table.

      i) DP_GET_CLIENT_TABLE generates a reference to the global object LocalTables.

      ii) The URL is passed to the LocalTables object and returns a reference to the data object.

      iii) The data object is copied to the internal table passed to DP_GET_CLIENT_TABLE.

      iv) The reference to the data object is deleted from the LocalTables object and released.

   c) As result, DP_GET_STREAM returns the data, the MIME type and the size of the BLOB data in bytes.

Other Data Storage Options

You can also store data on ftp:, http: or file: servers. To do this, all you have to do is assign the data and call the method SaveDataToUrl in the Data Provider object.

Since these or similar procedures for sending and receiving data are required again and again, the most important steps are encapsulated in ABAP function modules.

Function Modules

ABAP Function Modules used by the SAP Data Provider

Function module	Description
DP_CONTROL_ASSIGN	Assigns a Data Provider object to a control.
DP_CONTROL_ASSIGN_STREAM	Assigns an untyped internal table to a control property.
DP_CONTROL_ASSIGN_TABLE	Assigns a typed internal table to a control property.
DP_CONTROL_ASSIGN_URL	Assigns a URL to a control property via the SAP Data Provider.
DP_CONTROL_GET	Assigns a control property to a Data Provider object.
DP_CONTROL_GET_STREAM	Assigns a control property to an untyped internal table
DP_CONTROL_GET_TABLE	Assigns a control property to a typed internal table.
DP_CREATE	Creates a Data Provider object.

DP_CREATE_FROM_STREAM	Creates a Data Provider object from an untyped internal table.
DP_CREATE_FROM_TABLE	Creates a Data Provider object from a typed internal table.
DP_CREATE_URL	Creates a reference in the LocalTables data pool pointing to an untyped internal table.
DP_CREATE_URL_FROM_TABLE	Creates a reference in the LocalTables data pool pointing to a typed internal table.
DP_DESTROY	Destroys a Data Provider object.
DP_GET_FIELDS_FROM_TABLE	Returns the structure RFC_FIELDS from an internal table.
DP_GET_OBJECT	Returns the automation object initialized in a Data Provider object.
DP_GET_STREAM	Reads a stream from a Data Provider object into an untyped internal table.
DP_GET_STREAM_FROM_URL	Reads the data of a URL into an untyped internal table.
DP_GET_TABLE	Reads a typed internal table from a Data Provider object.
DP_GET_TABLE_FROM_CLIENT	Reads a typed internal table from the LocalTables data pool.
DP_SEND_STREAM	Sends an untyped internal table to a Data Provider object.
DP_SEND_TABLE	Sends a typed internal table to a Data Provider object.
DP_SET_OBJECT	Assigns an object to a Data Provider object.
DP_SET_URL	Sets a URL in the Data Provider object or in the format object.

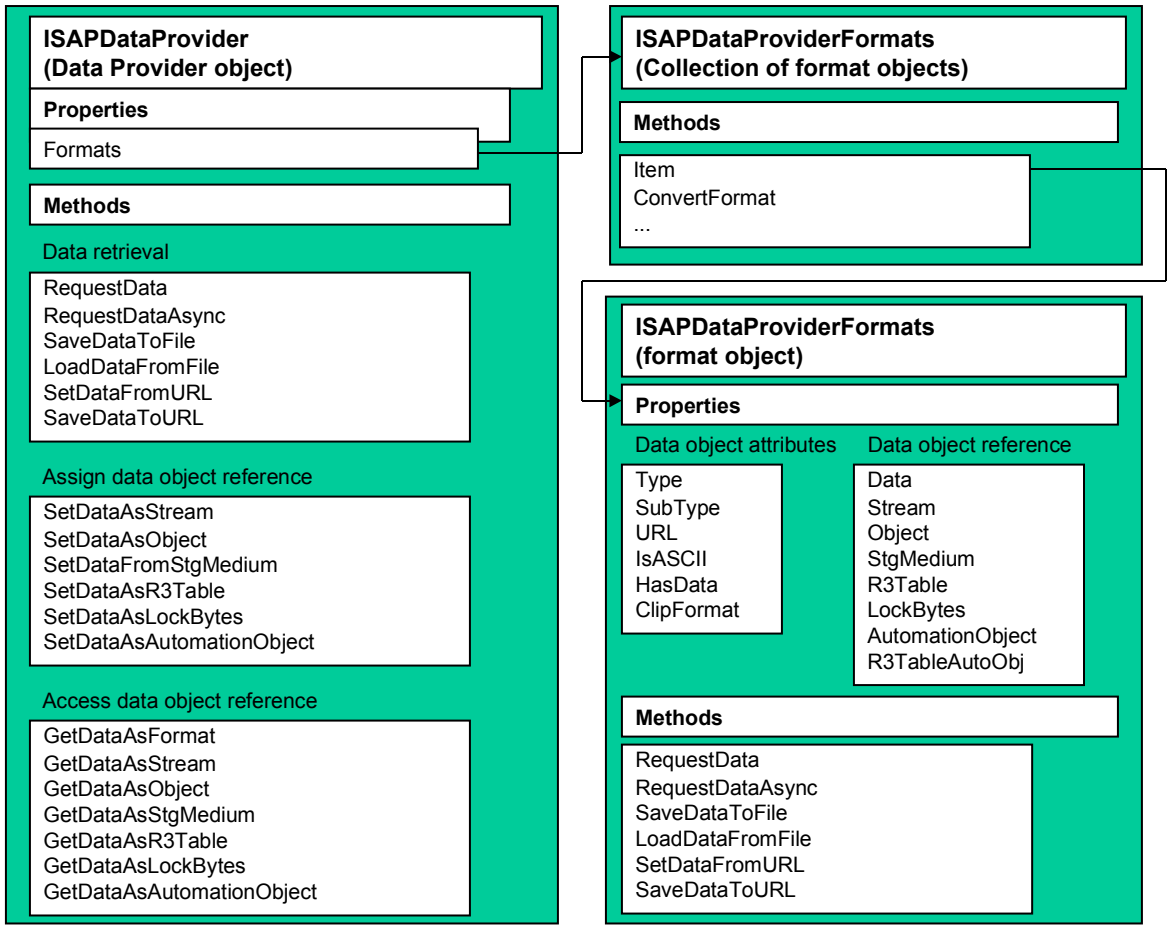
For detailed documentation, see the function group CNDP in the R/3 System.

Data Provider Object

General Function Description

Data Provider Object

Data Provider Object



The Data Provider object on the client side manages data in format objects, which you access via the collection *DataProvider.Formats*. Each format object is described either by its MIME type and the reference to a data object, or by a URL for retrieving the data.

The number of format objects or Data Provider objects involved depends on the data:

- If the client requires several logically identical data objects in different formats, there must be several format objects

  An example of this would be a text that exists as an rtf text and a WinWord document.

- If the client requires several data objects that are not logically identical, you must use several Data Provider objects

  This is no problem, since creating a Data Provider object requires very little runtime.

Creating New Format Objects

You create new format objects:

- Automatically by format conversion
- Explicitly

In the case of explicit assignment, the user is responsible for ensuring that the data is logically identical.

Since a format object can contain practically any data object, you cannot be certain that several format objects in a Data Provider object always contain identical data, because not every data object provides DataChange events.

Also, since performing a conversion after each DataChange would be a very expensive operation, format objects are only guaranteed to contain identical data immediately after calling *DataProvider.Formats.ConvertFormat*.

Accessing Data

You access data:

- Directly via the format object
- Through methods of the Data Provider object by specifying the desired format (MIME type).

SAP recommends that you always access data via the Data Provider object.

The Data Provider object and the format objects also contain other information that is described in detail in the interface documentation.

Data is retrieved either via a URL or by direct assignment of data. When retrieving data via a URL, you can either create a new format object or call the method *DataProvider.SetDataFromUrl*. You can also use the MIME type of the format objects to specify which MIME types are of interest.

Depending on requirements, you can request data:

- Synchronously
- Asynchronously

  In the case of asynchronous retrieval, the events of the Data Provider object inform the data user about the progress of the request.

Retrieval of data via http:, ftp:, file: and from the LocalTables object are transparent for the user. If you are addressing data from the LocalTables object, you must first use the function module DP_CREATE_URL to export the data from the ABAP program. It is stored in the client's main memory.

Sending Data

The Data Provider object allows you to send data to:

- A server

  To do this, you use the method *SaveDataToUrl* in the Data Provider object, or the format object. Any problems with write authorizations arising from this are the user's own responsibility.

- An ABAP program

To do this, you must specify a URL that does **not** begin with http:, ftp: or file: If no URL is specified, a unique URL („sapr3://<GUID>“) is generated.

The data addressed via this URL can be accessed by the ABAP program via the function modules DP_GET_TABLE, DP_GET_STREAM or DP_GET_OBJECT.

Standardized Data Retrieval

To standardize data retrieval, data objects retrieved by the Data Provider object in the client are always addressed via a URL with a unique name. At present, http:, ftp:, file: and sapr3: are supported (sapr3: refers to tables retrieved from an R/3 System by RFC).

To standardize the processing of data objects retrieved by the Data Provider object in the client as much as possible, MIME types must be assigned to them. For this reason, the Data Provider object only accepts data with a MIME type.
You assign the MIME type either when setting the data, or it is determined from the HTTP ResponseHeader or from the MIME type specified in the ABAP program.
At present, the MIME type „**application/x-unknown**“ is used for the namespaces ftp: and file:.
Data passed from an ABAP program must also be passed with a MIME type. For typed internal tables, the MIME type „**application/x-r3table**“ is used.

You can also assign data directly to the SAP Data Provider. Here, you do not use a URL, but assign the data via media (interfaces) known to the DataProvider object. With direct assignment of data, you must always specify the MIME type.

When using typed internal tables, do not use the MIME type „**application/x-r3table**“, because special function modules are available for this purpose. These generate and pass additional information that is required by this MIME type.

Standardized Data Representation

The SAP Data Provider exposes data objects in different forms known as **media**.

A medium exposes the data in a particular storage medium (interface), but the format and content remain the same. Most of the media used are standard COM interfaces, but SAP-specific interfaces are used for typed tables. Access to OLE DB rowsets is also possible.

The Data Provider object supports the following media:

Medium	Data representation	Remarks
Stream	IStream interface.	Corresponds to a byte stream with cursor.
LockBytes	ILockBytes interface.	Corresponds to a byte stream without cursor.
Object	IUnknown interface.	Corresponds to a COM object.

AutomationObject	IDispatch interface.	Corresponds to an OLE automation object. **CAUTION:** The AutomationObject depends on the MIME type. Usually, the methods and properties of the object represented via IDispatch vary considerably for each format object. The default implementation supports AutomationObject only for OLE documents of an OLE Automation server. For other types, this depends on the medium conversion object. To find out the options provided by an AutomationObject for conversion, refer to the conversion object documentation.
R3Table	Internal table. Data access is via the ISAPDPR3Table interface, which returns the ISAPrfclTab interface together with the rowset interfaces necessary for OLE DB.	You access the OLE DB rowset via the ISAPrfclTab interface or by converting the format objects from **application/x-r3table** to **application/x-rowset**. The conversion is easier and does not involve copying the data. In Visual Basic, this is also the only way you can access a RecordSet.
STGMEDIUM	STGMEDIUM structure.	This structure is not an interface, but can also be managed via reference counting.
R3TableAutoObj	IDispatch interface.	This automatable interface has the methods of the ISAPDPR3Table interface.
Data	VARIANT.	Always contains a copy of the data. **CAUTION:** Data depends on the MIME type. Usually, data stored in the VARIANT has many different format objects.
File	The data can be written to or read from a file.	

With direct assignment of data, you must use identical media:

- For most media, assignment is very easy, because you only have to assign a reference
- In the case of the media Data and File, you have to create a copy of the data in the format object of the Data Provider object.

Not all media are supported for all formats, and the default conversions do not support the conversion of all media. To find out the available media for a particular format, refer to the documentation of the medium conversion object.

Technical Reference

ISAPDataProvider

```
interface ISAPDataProvider : IDispatch
```

```
HRESULT get_PreferedFormat([out, retval] BSTR *)
```

Not implemented.

```
HRESULT put_PreferedFormat([in ] BSTR)
```

Not implemented.

```
HRESULT get_DefaultConsumer([out, retval] VARIANT *retval)
```

Not implemented.

```
HRESULT put_DefaultConsumer([in ] VARIANT Consumer)
```

Not implemented.

```
HRESULT get_Formats([out, retval] ISAPDataProviderFormats **retval)
```

Returns the current collection of format objects in the Data Provider object as the ISAPDataProviderFormats interface in *retval*.

The caller is responsible for releasing the reference to the parameter *retval*.

```
HRESULT get_Consumers([out, retval] ISAPDataProviderConsumers **retval)
```

Returns the current collection of registered consumers in the Data Provider object.

All objects in this collection receive the events of the Data Provider. To register a consumer in this collection, the Data Provider must use the standard OLE mechanisms for event handling (see OLE2 SDK, IConnectionPoint, IConnectionPointContainer).

```
HRESULT get_IsModified([out, retval] BOOL *retval)
```

Not implemented.

```
HRESULT RequestData([in] BSTR Type,[in] BSTR SubType, [in] VARIANT
Consumer)
```

Starts a data transfer for the specified MIME type (*Type/SubType*).

To enable data transfer, there must be a suitable format object in the format collection, and the format object must contain a valid URL.
The data retrieval process is synchronous. If this process is successful, the data can be requested from the format object. The consumer and all objects in the consumer collection are informed of the progress and the success of the data transfer through events. If data already exists in the specified format, it is overwritten by the new data.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
Consumer	Additional object interested only in the events of this request.

```
HRESULT RequestDataAsync(     [in] BSTR Type,[in] BSTR SubType,
[in] VARIANT Consumer)
```

Starts an asynchronous data transfer for the specified MIME type (*Type/SubType*).

To enable the data transfer, there must be a suitable format object in the format collection, and the format object must contain a valid URL.
The data retrieval process is asynchronous. Even if this process is successful, data cannot be requested from the format object until the relevant DataReady event is triggered. The consumer and all objects in the consumer collection are informed of the progress and the success of the data transfer through events. If data already exists in the specified format, it is overwritten by the new data.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
Consumer	Additional object interested only in the events of this request.

```
HRESULT AbortRequest(   [in] BSTR Type,
[in] BSTR SubType)
```

Terminates an asychnronous data transfer for the specified MIME type (*Type/SubType*).

```
HRESULT GetLastChange(  [in] BSTR Type,[in] BSTR SubType, [out,retval]
VARIANT *Delta)
```

Not implemented.

```
HRESULT SaveDataToFile([in] BSTR Type, [in] BSTR SubType, [in] BSTR
FileName)
```

Saves the data of the suitable format object according to the *Type/SubType* in the file specified in *FileName*.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
FileName	Path to file where data is to be written.

```
HRESULT GetDataAsStream(     [in] BSTR Type, [in] BSTR SubType,
[out,retval ] IStream **Stream)
```

Returns the data of the suitable format object according to the *Type/SubType* as an IStream interface.

The caller is responsible for releasing *Stream*.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
Stream	IStream interface to data in relevant format object.

```
HRESULT GetDataAsObject(     [in] BSTR Type, [in] BSTR SubType,
[out,retval ] IUnknown **Object)
```

Returns the data of the suitable format object according to the *Type/SubType* as an IUnknown interface.

The caller is responsible for releasing *Object*.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
Object	IUnknown interface to data in relevant format object.

```
HRESULT GetDataAsStgMedium(  [in] BSTR Type, [in] BSTR SubType,
[in] DWORD DesiredTyMed,
```

```
[out,retval ] STGMEDIUM *StgMedium)
```

Returns the data of the suitable format object according to the *Type/SubType* as STGMEDIUM.

DesiredTymed specifies which part of the structure *StgMedium* is to be filled.

The caller is responsible for releasing *StgMedium* via the OLE2 API. The STGMEDIUM returned by the Data Provider object usually has the element punkForRelease set in *StgMedium*. Therefore, releasing single elements within *StgMedium* with corresponding Windows APIs is **not** allowed.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
DesiredTymed	Path to file where data is to be written.
StgMedium	STGMEDIUM structure that contains the data.

```
HRESULT GetDataAsR3Table(     [in] BSTR Type, [in] BSTR SubType,
[out,retval ] ISAPDPR3Table **Table)
```

Returns the data of the suitable format object according to the *Type/SubType* as ISAPDPR3Table.

The caller is responsible for releasing *Table*.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
Table	ISAPDPR3Table interface for direct access to the internal table.

```
HRESULT GetDataAsR3TableExt(  [in] BSTR Type, [in] BSTR SubType,
[out, retval] VARIANT *vaR3Table)
```

Returns the data of the suitable format object according to the *Type/SubType* as VARIANT of the type VT_DISPATCH.

This IDispatch is part of the ISAPDPR3Table The caller is responsible for releasing *vaR3Table* via VariantClear.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.

SubType	MIME type of data object. Optional.
vaR3Table	Variant that contains the interface as VT_DISPATCH.

```
HRESULT GetDataAsFormat(      [in] BSTR Type, [in] BSTR SubType,
[out, retval] ISAPDataProviderFormat **FormatObj)
```

Returns the data of the suitable format object according to the *Type/SubType* as ISAPDataProviderFormat.

The caller is responsible for releasing *FormatObj*.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
FormatObj	ISAPDataProviderFormat interface for direct access to the format object.

```
HRESULT LoadDataFromFile(     [in] BSTR Type, [in] BSTR SubType,
[in] BSTR FileName)
```

Reads the data from the file *FileName* into the suitable format object according to the *Type/SubType*.

You must first create the format object via the format collection.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
FileName	Path and filename to be read.

```
HRESULT SetDataAsStream(      [in] BSTR Type, [in] BSTR SubType,
[in ] IStream *Stream)
```

Places the data passed via *Stream* in the format object (*Type/SubType*).

If the format object does not exist, a new format object is created. All references in the format object to other data objects are released.

Parameters

Parameter	Description
Type	MIME type of data object. This parameter is **mandatory**.
SubType	MIME type of data object. This parameter is **mandatory**.

Stream	IStream interface that contains the new data.

```
HRESULT SetDataAsObject(      [in] BSTR Type, [in] BSTR SubType,
[in ] IUnknown *Object)
```

Places the data passed via *Object* in the format object (*Type/SubType*).

If the format object does not exist, a new format object is created. All references in the format object to other data objects are released.

Parameters

Parameter	Description
Type	MIME type of data object. This parameter is **mandatory**.
SubType	MIME type of data object. This parameter is **mandatory**.
Object	IUnknown interface to any object.

```
HRESULT SetDataFromStgMedium( [in] BSTR Type, [in] BSTR SubType,
[in ] STGMEDIUM *StgMedium)
```

Places the data passed via *StgMedium* in the format object (*Type/SubType*).

If the format object does not exist, a new format object is created. All references in the format object to other data objects are released. In *StgMedium*, punkForRelease must **not** be set. The format object installs its own punkForRelease in *StgMedium* to control the life of the data

Parameters

Parameter	Description
Type	MIME type of data object. This parameter is **mandatory**.
SubType	MIME type of data object. This parameter is **mandatory**.
StgMedium	STGMEDIUM with the new data.

```
HRESULT SetDataAsR3Table([in ] ISAPDPR3Table *Table)
```

Places the data passed via *Table* in the format object (*Type/SubType*).

If the format object does not exist, a new format object is created. All references in the format object to other data objects are released. Since *Type/SubType* is part of the ISAPDPR3Table interface, these parameters do not have to be specified.

```
HRESULT SetDataAsR3TableExt([in ] VARIANT vaR3Table)
```

Places the data passed via *vaR3Table* in a format object of the format *Type/SubType* according to *vaR3Table*.

If the format object does not exist, a new format object is created. All references in the format object to other data objects are released. Since *Type/SubType* is part of the ISAPDPR3Table

interface, these parameters do not have to be specified. *VaR3Table* must be a variant of the type VT_DISPATCH, which contains the IDispatch part of an ISAPDPR3Table interface.

```
HRESULT SetDataFromURL([in] BSTR Type, [in] BSTR SubType, [in] BSTR
URL)
```

Places the data addressed via *URL* in the suitable format object (*Type/SubType*).

You do not have to create the format object first via the format collection. If a format object has been created, it is used. Otherwise, a format object is created depending on the MIME type of the data. If the data in the URL does not match the specified MIME type, no data is passed.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
Url	URL pointer to the data. Supported namespaces include ftp:, http:, and sapr3:.

```
HRESULT SaveDataToURL(  [in] BSTR Type, [in] BSTR SubType,
[in] BSTR DesiredURL, [out, retval] BSTR *URL)
```

Saves the data of the suitable format object in the specified *DesiredURL*.

The format object must be created first via the format collection, and it must contain data. If no *DesiredUR*L is specified, a URL of the form „sapr3://<GUID>“ is created and returned as *URL*. This can then be loaded from an ABAP program. The caller is responsible for releasing *Url* with sysFreeString.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
DesiredUrl	URL pointer to the data. Supported namespaces include ftp:, http:, and sapr3:.
Url	URL pointer to the data. Supported namespaces include ftp:, http:, and sapr3:.

```
HRESULT GetDataAsLockBytes(   [in] BSTR Type, [in] BSTR SubType,
[out,retval ] ILockBytes **LockBytes)
```

Returns the data of the suitable format object according to *Type/SubType* as ILockBytes.

The caller is responsible for releasing *LockBytes*.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
LockBytes	ILockBytes interface to data in relevant format object.

```
HRESULT SetDataAsLockBytes(  [in] BSTR Type, [in] BSTR SubType,
[in ] ILockBytes *LockBytes)
```

Places the data passed via *LockBytes* in the format object (*Type/SubType*).

If the format object does not exist, a new format object is created. All references in the format object to other data objects are released.

Parameters

Parameter	Description
Type	MIME type of data object. This parameter is **mandatory**.
SubType	MIME type of data object. This parameter is **mandatory**.
LockBytes	ILockBytes interface that contains the new data. See OLE2 reference.

```
HRESULT SetDataAsAutomationObject( [in] BSTR Type, [in] BSTR SubType,
[in ] Idispatch *Object)
```

Places the automation object passed via *Object* in the format object (*Type/SubType*).

If the format object does not exist, a new format object is created. All references in the format object to other data objects are released.

The medium AutomationObject represented via IDispatch interface depends on the MIME type of the data and can only result from a format or medium conversion. To find out which object is returned here, refer to the documentation of the relevant conversion object. For typed internal tables, the MIME type **application/x-r3table** is always used. The AutomationObject of this MIME type is the SAP TableObject from the SAP TableFactory (see RFC SDK documentation). Internal tables can also be addressed via the MIME type **application/x-rowset**. In this case, a RecordSet object (see OLE DB reference) is used as AutomationObject. To generate an object of the type **application/x-rowset** from an object of the type **application/x-r3table**, you must use the method ISAPDataProviderFormats::ConvertFormat of the format collection.

Parameters

Parameter	Description

Type	MIME type of data object. This parameter is **mandatory**.
SubType	MIME type of data object. This parameter is **mandatory**.
Object	IDispatch interface that represents the new object. See OLE2 reference.

```
HRESULT GetDataAsAutomationObject( [in] BSTR Type, [in] BSTR SubType,
[out, retval] IDispatch **Object)
```

Returns the automation object within the suitable format object according to *Type/SubType* as IDispatch.

The caller is responsible for releasing *Object*.

Parameters

Parameter	Description
Type	MIME type of data object. Optional.
SubType	MIME type of data object. Optional.
Object	IDispatch interface to automation object in relevant format object.

ISAPDataProviderFormats

```
interface ISAPDataProviderFormats : IDispatch


HRESULT Add(      [in] BSTR Type,
                  [in] BSTR SubType,
                  [in] BSTR Url,
                  [in] BSTR Description,
                  [in] BSTR Server,
                  [out, retval] ISAPDataProviderFormat **retval)
```

Adds a new format object to the collection and returns the new object in *retval*.

The caller is responsible for releasing the reference to *retval*.

If no *Type/SubType* is specified, this is determined during the data transfer (with RequestData). If *Type/SubType* is specified, only data of this type is requested and processed during a data transfer.

Parameters

Parameter	Description
Type	MIME type of new format object. Optional.
SubType	MIME type of new format object. Optional.
Url	URL pointer to data of new format object.

Description	Additional description of format object. This does not affect other processing steps.
Server	Name of machine where data originates. Optional.
RetVal	ISAPDataProviderFormat interface to new format object.

```
HRESULT Remove(   [in] BSTR FormatName,
[in] BSTR SubFormatName,
[out, retval] BOOL *retval)
```

Deletes the suitable format object from the collection.

Retval is TRUE if a suitable format object is found.

Parameters

Parameter	Description
Type	MIME type of format object to be deleted. Optional.
SubType	MIME type of format object to be deleted. Optional.
RetVal	TRUE if format object could be deleted.

```
HRESULT RemoveAll()
```

Deletes all format objects from the collection.

```
HRESULT get_Item( [in] BSTR FormatName, [in] BSTR SubFormatName,
[out, retval] ISAPDataProviderFormat **retval)
```

Returns the suitable format object according to *Type/SubType* from the collection.

The caller is responsible for releasing the reference to *retval*.

Parameters

Parameter	Description
Type	MIME type of format object. Optional.
SubType	MIME type of format object. Optional.
RetVal	ISAPDataProviderFormat interface to format object.

```
HRESULT ConvertFormat(  [in] BSTR Type,                    [in] BSTR
SubType,                    [in] BSTR NewType,
      [in] BSTR NewSubType,                    [in] SapDPMediumType
NewMedium,                  [in] BOOL DoAutoConversion,
      [in] VARIANT Consumer,                   [out, retval]
ISAPDataProviderFormat **retval);
```

Technical Reference

Converts the suitable format object (*Type/SubType*) to a new format object *NewType/NewSubType*).

As a prerequisite, a format conversion object must be installed. *NewMedium* specifies the preferred medium for the new format object after the conversion. The new format object is added to the collection and returned as *retval*. The caller is responsible for releasing the reference to *retval*.

Parameters

Parameter	Description
Type	MIME type of source format object for conversion.
SubType	MIME type of source format object for conversion.
NewType	MIME type of target format object for conversion.
NewSubType	MIME type of target format object for conversion.
NewMedium	Preferred medium to be provided by new format object.
DoAutoConversion	Controls automatic conversions if source format object gets new data. If this parameter is TRUE, ConvertFormat is called again when a new data object is placed in the source format object.
Consumer	Additional event receiver only for these conversion functions.
RetVal	ISAPDataProviderFormat interface of the new format object.

```
HRESULT ConvertFormatAsync(   [in] BSTR Type,
      [in] BSTR SubType,                              [in] BSTR NewType,
                            [in] BSTR NewSubType,
      [in] BOOL DoAutoConversion,                          [in] VARIANT
Consumer,                          [out, retval] ISAPDataProviderFormat
**retval)
```

Creates a new format object (*NewType/NewSubType)* and returns this as *retval*.

The caller is reponsible for releasing the reference to *retval*. The data is not converted to the new format object until it has been made available for the suitable format object (*Type/SubType)*. Otherwise, this method behaves the same as ConvertFormat.

```
HRESULT _NewEnum([out, retval] LPUNKNOWN* retval)
```

Returns the standard OLE enumerator of the collection.

```
HRESULT Force(    [in] BSTR Type,              [in] BSTR SubType,
            [in] BSTR Url,               [in] BSTR Description,
      [in] BSTR Server,            [out, retval] ISAPDataProviderFormat
**retval)
```

Forces the suitable format object according to *Type/SubType*.

If a suitable format object already exists in the collection, this is returned in *retVal*. Otherwise, a new format object is created and returned in *retVal*. The caller is responsible for releasing the reference to *retval*.

Parameters

Parameter	Description
Type	MIME type of new format object. Optional.
SubType	MIME type of new format object. Optional.
Url	URL pointer to data of new format object.
Description	Additional description of format object. This does not affect other processing steps.
Server	Name of machine where data originates. Optional.
RetVal	ISAPDataProviderFormat interface to new format object.

```
HRESULT ConvertByClipFormat(  [in] BSTR Type,
                              [in] BSTR SubType,
                              [in] long cf,
                              [in] SapDPMediumType NewType,
                              [in] BOOL DoAutoConversion,
                              [in] VARIANT Consumer,
                              [out, retval] ISAPDataProviderFormat
**retval)
```

Converts the suitable format object according to *Type/SubType* to a new format object.

The new format object must support the specified clipboard format. *NewMedium* specifies the preferred medium for the new format object after the conversion.

Parameters

Parameter	Description
Type	MIME type of source format object for conversion.
SubType	MIME type of source format object for conversion.
Cf	Clipboard format of target format object.
NewType	Preferred medium to be provided by new format object.
DoAutoConversion	Controls automatic conversions if source format object gets new data.
Consumer	Additional event receiver only for these conversion functions.
RetVal	ISAPDataProviderFormat interface of the new format object.

ISAPDataProviderFormat

```
interface ISAPDataProviderFormat : IDispatch
```

```
HRESULT get_Data([out, retval] VARIANT *retval)
```

Returns the data of the format object as VARIANT.

The contents of *retval* depend on the MIME type of the format object. The caller is repsonsible for releasing *retval* with VariantClear.

```
HRESULT put_Data([in] VARIANT vaData)
```

Places the data stored in *vaData* in the format object.

In most cases, you cannot distinguish whether the data corresponds to the current MIME type, so the caller is responsible for using data of the right type. The contents of *vaData* depend on the MIME type of the format object. All references that are stored in the format object and point to other data objects are released.

```
HRESULT get_IsUpdateable([out, retval] BOOL *retval)
```

Returns TRUE if the data can be changed in the format object.

```
HRESULT get_IsCacheable([out, retval] BOOL *retval)
```

Returns TRUE if the data can be stored as persistent in the client.

```
HRESULT get_Type([out, retval] BSTR *retval)
```

Returns the type of the current format object.

The caller is responsible for releasing *retVal* via SysFreeString.

```
HRESULT get_IsASCII([out,retval] BOOL  *retval)
```

Returns TRUE if the data in the format object consists of only alphanumeric characters.

```
HRESULT get_SubType([out, retval] BSTR *retval)
```

Returns the SubType of the current format objects.

```
HRESULT get_URL([out, retval] BSTR *retval)
```

Returns the URL used to retrieve the data.

If the data was assigned directly, using one of the SetDataAs… methods, a blank string is returned in *retval*.

```
HRESULT get_Description([out, retval] BSTR *retval)
```

Returns an additional description of the format object in *retval*.

The caller is responsible for releasing *retVal* via SysFreeString.

```
HRESULT get_NeedsConversion([out, retval] BOOL *retval)
```

Returns TRUE if the format object has resulted from a conversion and a new data object was assigned to the source format object of the conversion in the meantime.

```
HRESULT get_HasData([out, retval] BOOL *retval)
```

Returns TRUE if the format object contains data.

```
HRESULT get_AutoConversion([out, retval] BOOL *retval)
```

Returns TRUE if the format has resulted from a conversion with *bDoAutoConversion* = *TRUE*.

```
HRESULT put_AutoConversion([in ] BOOL NewVal)
```

Sets the AutoConversion flag of the format object to the new value.

```
HRESULT get_ClipFormat([out, retval] CLIPFORMAT *retval)
```

Returns the clipboard format of the format object.

The clipboard format depends on the MIME type of the format object, and is stored in the registry for different MIME types.

```
HRESULT put_ClipFormat([in] CLIPFORMAT retval)
```

Assigns a clipboard format to the format object.

```
HRESULT put_IsUpdateable([in ] BOOL NewVal)
```

Sets the property IsUpdateable to *NewVal*.

If IsUpdateable has the value TRUE, the data of the format object can be modified.

```
HRESULT put_IsCacheable([in ] BOOL NewVal)
```

If IsCacheable is set to TRUE, the data of the format object can be stored as persistent in the client.

```
HRESULT put_Description([in ] BSTR NewDesc)
```

Inserts the specified description *NewDesc* into the format object.

This description is not a semantic meaning.

```
HRESULT RequestData([in] VARIANT Consumer)
```

If the property URL of the format object contains a valid URL, the data to which the URL points is loaded synchronously and stored in the format object.

Events inform all consumers of the Data Provider object and the specifed *Consumer* about the progress and the success of the data transfer.

```
HRESULT RequestDataAsync([in] VARIANT Consumer)
```

If the property URL of the format object contains a valid URL, the data to which the URL points is loaded asynchronously and stored in the format object.

Events inform all consumers of the Data Provider object and the specifed *Consumer* about the progress and the success of the data transfer.

```
HRESULT SaveDataToFile([in] BSTR FileName)
```

Saves the data stored in the format object to the file specified in *FileName*.

```
HRESULT LoadDataFromFile([in] BSTR FileName)
```

Loads the data saved in the file specified in *FileName* into the format object.

```
HRESULT get_Stream([out,retval ] IStream **Stream)
```

Returns the data of the format object in *Stream*.

The caller is responsible for releasing the reference.

```
HRESULT get_Object( [out,retval ] IUnknown **Object)
```

Returns the object of the format object in *Object*.

The caller is responsible for releasing the reference to *Object*.

```
HRESULT get_AutomationObject( [out,retval ] IDispatch **Object)
```

Returns the AutomationObject of the format object in *Object*.

The caller is responsible for releasing the reference to *Object*. Whether an IDispatch is derived from the data of the format object depends on the MIME type of the format object.

```
HRESULT GetDataAsStgMedium(   [in] DWORD DesiredTyMed,
[out,retval ] STGMEDIUM *StgMedium)
```

Returns the data of the format object in *StgMedium*.

The caller is reponsible for releasing the reference to ReleaseStgMedium (see OLE2 API). The desired TYMED (see OLE2 API) is specifed in *DesiredTyMed*. You can link different TYMED objects with a |.

```
HRESULT get_R3Table([out,retval ] ISAPDPR3Table **Table)
```

Returns the data of the format object as ISAPDPR3Table.

The caller is responsible for releasing the reference. Part of the ISAPDPR3Table interface is the ISAPrfclTab interface. This allows you access to the internal table.

```
HRESULT putref_Stream([in ] IStream *Stream)
```

Stores the reference to *Stream* in the format object.

All references that are stored in the format object and point to other data objects are released. The format object holds this reference until new data is assigned or the format object itself is released. Since the MIME type of the stream cannot usually be determined, the user is responsible for specifying a *Stream* that corresponds to the current MIME type.

```
HRESULT putref_Object([in ] IUnknown *Object)
```

Stores the reference to *Object* in the format object.

All references that are stored in the format object and point to other data objects are released. The format object holds this reference until new data is assigned or the format object itself is released. Since the MIME type of the object cannot usually be determined, the user is responsible for specifying an *Object* that corresponds to the current MIME type.

```
HRESULT SetDataFromStgMedium([in ] STGMEDIUM *StgMedium)
```

Fills the format object with the data from the *StgMedium*.

All references that are stored in the format object and point to other data objects are released. The format object holds this reference until new data is assigned or the format object itself is released. Since the MIME type of the *Object* cannot usually be determined, the user is responsible for specifying a *StgMedium* that corresponds to the current MIME type.

```
HRESULT putref_R3Table([in ] ISAPDPR3Table *Table)
```

Stores the reference to *Table* in the format object.

All references that are stored in the format object and point to other data objects are released. The format object holds this reference until new data is assigned or the format object itself is released. Since the MIME type is stored in ISAPDPR3Table, data with the wrong MIME type is rejected.

```
HRESULT SetDataFromURL([in ] BSTR Url)
```

Reads the data of the specified URL and stores it in the format object.

All references that are stored in the format object and point to other data objects are released. The format object holds this reference until new data is assigned or the format object itself is released. Data with an unsuitable MIME type is rejected

```
HRESULT SaveDataToURL([in ] BSTR DesiredUrl,[out, retval] BSTR *Url)
```

Transports the data in the format object to *DesiredUrl*.

If no *DesiredUrl* is specified, the data is stored in the global LocalTables object, and a unique URL is generated. This URL is returned in *URL* to the caller. The caller is responsible for releasing *Url* via SysFreeString.

```
HRESULT putref_AutomationObject( [in ] IDispatch *Object)
```

Stores the reference to *Object* in the format object.

All references that are stored in the format object and point to other data objects are released. The format object holds this reference until new data is assigned or the format object itself is released. Since the MIME type of the IDispatch cannot usually be determined, the user is responsible for specifying an *Object* that corresponds to the current MIME type.

```
HRESULT putref_LockBytes([in] ILockBytes *LockBytes)
```

Stores the reference to *LockBytes* in the format object.

All references that are stored in the format object and point to other data objects are released. The format object holds this reference until new data is assigned or the format object itself is released. Since the MIME type of the ILockBytes cannot usually be determined, the user is responsible for specifying a *LockBytes* that corresponds to the current MIME type.

```
HRESULT get_LockBytes([out,retval ] ILockBytes **LockBytes)
```

Returns the data of the format object in L*ockBytes*.

The caller is responsible for releasing the reference.

```
HRESULT putref_UserObj([in] IUnknown *AnyObj)
```

You can attach any object to the format object via the property UserObj.

This call releases existing references to a UserObj.

```
HRESULT get_UserObj( [out, retval] IUnknown **AnyObj)
```

Returns the UserObj attached to the format object.

The caller is responsible for releasing the reference to *AnyObj*.

```
HRESULT get_R3TableAutoObj( [out, retval] IDispatch **AnyObj)
```

Returns the IDispatch interface of the ISAPDPR3Table interface if data exists in the medium R3Table.

```
HRESULT putref_R3TableAutoObj( [in ] IDispatch *Object)
```

The *Object* must be the IDispatch of an ISAPDPR3Table interface.

```
HRESULT Clone ([in] BSTR Range, [out,retval] ISAPDataProviderFormat
**ppRet)
```

Creates a new format object from the current format object and returns it in *ppRet*.

The caller is responsible for releasing the reference. The new format object is not inserted into the format collection. The parameter Range describes that part of the format object to be returned as Clone. The standard implementation does not support this function. Clone objects can only be generated by medium conversion objects and therefore depend on the MIME type of the format object.

```
HRESULT Range([out,retval ] BSTR *Range)
```

Returns the range of the format object in *Range*.

The caller must release *Range* via SysFreeString. A format object only has a range if it is generated via the method Clone.

```
HRESULT put_IsASCII([in] BOOL  newVal)
```

Sets the property IsASCII to TRUE.

ISAPDataProviderEvents

```
interface ISAPDataProviderEvents : IDispatch
```

```
HRESULT OnStart([in] BSTR Type,[in] BSTR SubType)
```

This event is triggered whenever a format object begins to retrieve data of the format *Type/SubType* that was addressed via its URL.

Parameters

Parameter	Description
Type	MIME type of format object.
SubType	MIME type of format object.

```
HRESULT OnProgress([in] BSTR Type,[in] BSTR SubType, [in] USHORT
PerCentDome)
```

This event is triggered whenever a format object retrieves data of the format *Type/SubType* that was addressed via its URL.

PerCentDone specifies how much of the process has been completed. The value of *PerCentDone* can be anywhere in the range from 0 to 100.

Parameters

Parameter	Description
Type	MIME type of format object.
SubType	MIME type of format object.
PerCentDone	Indicates progress of load process.

```
HRESULT OnDone([in] BSTR Type,[in] BSTR SubType, [in] SCODE ErrorCode)
```

This event is triggered whenever a format object concludes data retrieval of the format *Type/SubType* that is addressed via its URL.

It is irrelevant whether the data retrieval has been successful or not.

Parameters

Parameter	Description
Type	MIME type of format object.
SubType	MIME type of format object.
ErrorCode	0: Successful data transfer. Other: See OLE2 or INETSDK ErrorCodes.

```
HRESULT OnDataReady([in] BSTR Type,[in] BSTR SubType)
```

This event is triggered whenever data of the format *Type/SubType* is available in the format object.

This can depend both on a synchronous or an asynchronous load process, as well as a direct assignment via one of the SetDataAs… methods.

Parameters

Parameter	Description
Type	MIME type of format object.
SubType	MIME type of format object.

```
HRESULT OnDataChange([in] BSTR Type,[in] BSTR SubType)
```

This event is triggered whenever **new** data objects of the format *Type/SubType* are available in the format object.

This can depend both on a synchronous or an asynchronous load process, as well as a direct assignment via one of the SetDataAs… methods. This event is identical to the OnDataReady event.

No events are triggered if the data changes within a data object. Here, you must ensure that all possible data objects generate appropriate events. However, since this is not always the case, you cannot always rely on these events occurring.

Parameters

Parameter	Description
Type	MIME type of format object.
SubType	MIME type of format object.

```
HRESULT OnConversionStart([in] BSTR Type,[in] BSTR SubType)
```

This event is triggered before the conversion of a format object to a new format object of the format *Type/SubType*.

Parameters

Parameter	Description
Type	MIME type of format object.
SubType	MIME type of format object.

```
HRESULT OnConversionDone([in] BSTR Type,[in] BSTR SubType,[in] SCODE
ErrorCode)
```

This event is triggered after the conversion of a format object to a new format object of the format *Type/SubType*.

Parameters

Parameter	Description
Type	MIME type of format object.
SubType	MIME type of format object.
ErrorCode	0: Successful data transfer. Other: See OLE2 or INETSDK ErrorCodes.

ISAPDPR3Table

The ISAPDPR3Table interface is used mainly to encapsulate internal tables. However, you cannot refer back to the ISAPrfcITab interface here, because the DataProvider object has the following additional requirements:

- The interface must be addressable via OLE Automation, so that it can also be addressed from ABAP programs or assigned to other objects.

- Besides the actual data, format objects of the DataProvider object have additional attributes like RFC_FIELDS table that describes typed tables, MIME types, size of the data object, system information about caching options and updates, etc.
- Format objects of the format **application/x-r3Table** need an additional description of the table type (schema information).

```
interface ISAPDPR3Table : IDispatch
```

```
HRESULT get_SubType([out,retval ] BSTR  *retval)
```

Returns the data type.

The caller is responsible for releasing *retval* via SysFreeString.

```
HRESULT get_SubType([out,retval ] BSTR  *retval)
```

Returns the data subtype.

The caller is responsible for releasing *retval* via SysFreeString.

```
HRESULT get_Date([out,retval ] BSTR  *retval)
```

Returns the date of the data.

The caller is responsible for releasing *retval* via SysFreeString. By default, the date is the current date, but the application programmer can change this value to specify the date of the last logical table change. This allows you to synchronize entries in a cache.

```
HRESULT get_Time([out,retval ] BSTR  *retval)
```

Returns the time of the data.

The caller is responsible for releasing *retval* via SysFreeString. By default, the time is the current time, but the application programmer can change this value to specify the date of the last logical table change. This allows you to synchronize entries in a cache.

```
HRESULT get_IsASCII([out,retval ] BOOL *retval)
```

Returns TRUE if the data in the table consists only of ASCII characters, FALSE if the table can contain any characters.

For untyped tables, get_IsASCII returns TRUE if the table contains a column of the ABAP data type „C“, otherwise the column is ABAP data type „X“.

```
HRESULT get_IsStructured([out,retval ] BOOL *retval)
```

Returns TRUE if the object is a typed table with field information.

```
HRESULT get_IsCacheable([out,retval ] BOOL *retval)
```

Returns TRUE if the table can be stored in a cache.

```
HRESULT get_Description([out,retval ] BSTR *retval)
```

Returns an additional description of the table.

This description has no special semantic meaning. The caller is responsible for releasing *retval* via SysFreeString.

```
HRESULT get_Platform([out,retval ] BSTR *retval)
```

Returns the operating system platform on which the table was created.

This statement is obsolete as of Release 4.5A. The caller is responsible for releasing *retval* via SysFreeString.

```
HRESULT get_Size([out,retval] ULONG *retval)
```

Returns the size of the data in the table (in bytes).

This value does not have to be the number of rows multiplied by the column width, because the last rows do not have to be completely filled.

```
HRESULT put_Size([in] ULONG Size)
```

Sets the size of the data in the table (in bytes).

If required, an appropriate number of lines can be generated or deleted.

```
HRESULT get_RowLength([out,retval] ULONG *retval)
```

Returns the width of the table.

```
HRESULT get_DataTable([out,retval] ISAPrfcITab **retval)
```

Returns the ISAPrfcITab interface of the table that contains the data.

```
HRESULT get_FormatTable([out,retval] ISAPrfcITab **retval)
```

Returns the ISAPrfcITab interface of the table that contains the description of the table structure.

The underlying table is always of the type RFC_FIELDS. If NULL is returned, it is a Binary Large Object (BLOB) data object with only one column of width RowLength, which is either ABAP data type „C“ (get_IsASCII == TRUE) or ABAP data type „X“.

```
HRESULT CopyDataTable([in] ISAPrfcITab *Table)
```

Copies the data in the table to the table *Table*.

In the case of typed tables, the table *Table* **must** correspond to the structure of the data, and is copied row by row. In the case of untyped tables, the column width of *Table* can differ from the column width of the data table of this object.

```
HRESULT CopyFormatTable([in] ISAPrfcITab *Table)
```

Copies the field description in the table to the table *Table*.

The table *Table* must be of the type RFC_FIELDS.

```
HRESULT get_IntRep([out,retval] SapDPIntRep *retval)
```

Returns the integer representation of the internal table.

As of Release 4.5A, this call is obsolete.

```
HRESULT CopyPropertiesTable([in] ISAPrfcITab *Table)
```

Not implemented.

```
HRESULT put_IsASCII([in ] BOOL newVal)
```

Sets the property IsASCII to *newVal*.

In the case of typed tables, this property is irrelevant. In the case of BLOB data, this property is used for any necessary codepage conversions and for conversion to an OLE VARIANT.

Standard Medium Conversions

The Data Provider format object allows standard conversions for common media. For certain MIME types, however, you can use medium conversion objects to overwrite the standard conversions.

Typical standard medium conversions are:

- Representation of internal tables as streams
- Access to the automation interface for OLE documents
- Reading of files
- Writing of files

SAP Data Provider Standard Conversions

FROM	TO	IStream	ILockBytes	IUnknown	IDispatch	ISA PDP R3T	STGMEDIUM
IStream	Yes	Yes	If stream is a compound document	If stream is a compound document and server allows OLE Automation	Yes	For untyped tables	
ILockBytes	Yes	Yes			Yes	No	

IUnknown	If IStream is implemented	No	Yes	If object allows OLE Automation	No	No
IDispatch		No	Yes	Yes	No	No
ISAPDPR3Table	Yes	Yes	If table contains a compound document	If table contains a compound document and server allows OLE Automation	Yes	For untyped tables
STGMEDIUM	Yes	Yes	No	No		Yes

Additionally, the standard conversion can generate IStream objects from files, or convert IStream objects to files. This means that every medium that can be converted to an IStream can be generated from a file, or written to a file.

The standard conversion to the data type VARIANT:

- If an IStream can be generated, a VARIANT of the type BSTR is generated if the property „IsASCII“ of the format objects is TRUE. Otherwise, a SAFEARRAY of VT_UI1 is generated.

  This operation **copies** the data.

- If the medium AutomationObject is supported in the format object, a VARIANT of the type VT_DISPATCH is generated.

  This operation copies only the reference as long as the AutomationObject has no default property. If a default property is implemented, it depends on the receiver of the data whether the AutomationObject or the default property is used.