Swyx Forum

Database Integration

for SwyxWare Extended Call Routing (VBScript based)

Tom Wellige

Contents

Preface 3

Preparation 4

Setup 5

Usage 8

How in include it into your own call routing 8

Via GSE Action 8

Via include Statement 9

Read Data 10

Edit Data 11

Add Data 12

Properties & Methods 13

IpPbx_Database 13

Property sVersion 13

Property nState 13

Property sState 13

Property nCommandTimeout 13

Property nLastError 13

Property sLastErrorDescr 14

Method Open 15

Method Close 15

Method NewQuery 15

Method ValidHandle 16

Method DisposeQuery 16

IpPbx_Query 17

Property sVersion 17

Property nState 17

Property sState 17

Property bEOF 17

Property bBOF 17

Property Fields 18

Property nLastError 18

Property sLastErrorDescr 19

Method Open 20

Method OpenReadOnly 21

Method OpenReadOnlyForwardOnly 21

Method OpenEdit 22
Method OpenEditBatch 22
Method OpenAdd 23
Method Close 23
Method Dump 24
Method MoveFirst 25
Method MovePrevious 25
Method MoveNext 26
Method MoveLast 27
Method Update 28
Method UpdateBatch 28
Examples 29
Example 1_NewQuery.vbs 29
Example 2_Open.vbs 29
Example 2_OpenAdd.vbs 29
Example 2_OpenEdit.vbs 29
Example 2_OpenEditBatch.vbs 29
Example 2_OpenReadOnly.vbs 29
Example 2_OpenReadOnlyForwardOnly.vbs 29
Example 3_Dump.vbs 29
Example 4_Field.vbs 29
Example 5_MoveNext.vbs 29
Example 6_Update.vbs 30
Example 6_UpdateBatch.vbs 30

Preface

This document describes the “Database Integration” for SwyxWares Call Routing, based on VBScript.

VBScript offers already database integration using the ADODB.Connection and ADODB.Recordset objects which can of course be fully utilised within the SwyxWare Call Routing.

However, things like error handling and writing some proper trace information for later problem analysis is fully in the hand of the developer of the VBScript code and might often times not meet proper standards.

Here comes the “Database Integration” into place. It not only makes certain aspects of database handling a little bit more easy, it also comes with full error handling and proper tracing. So a script developer can fully concentrate on his own code (at least in terms of database access).

The “Database Integration” encapsulates the ADODB.Connection and ADODB.Recordset objects and instead offers its functionality through an **IpPbx_Database** object.

Please note: you still have to use VBScript in your call routing script. This integration does not add specialised GSE Actions.

Preparation

Simply download the latest version of the “Database Integration” open source project.

Additionally you need to know how to connect to your desired database. This requires a so called “connection string”, you need to provide (like you would need to do when using the ADODB.Connection object).

A good source for examples of such connection strings for all types of databases and access types can be found here:

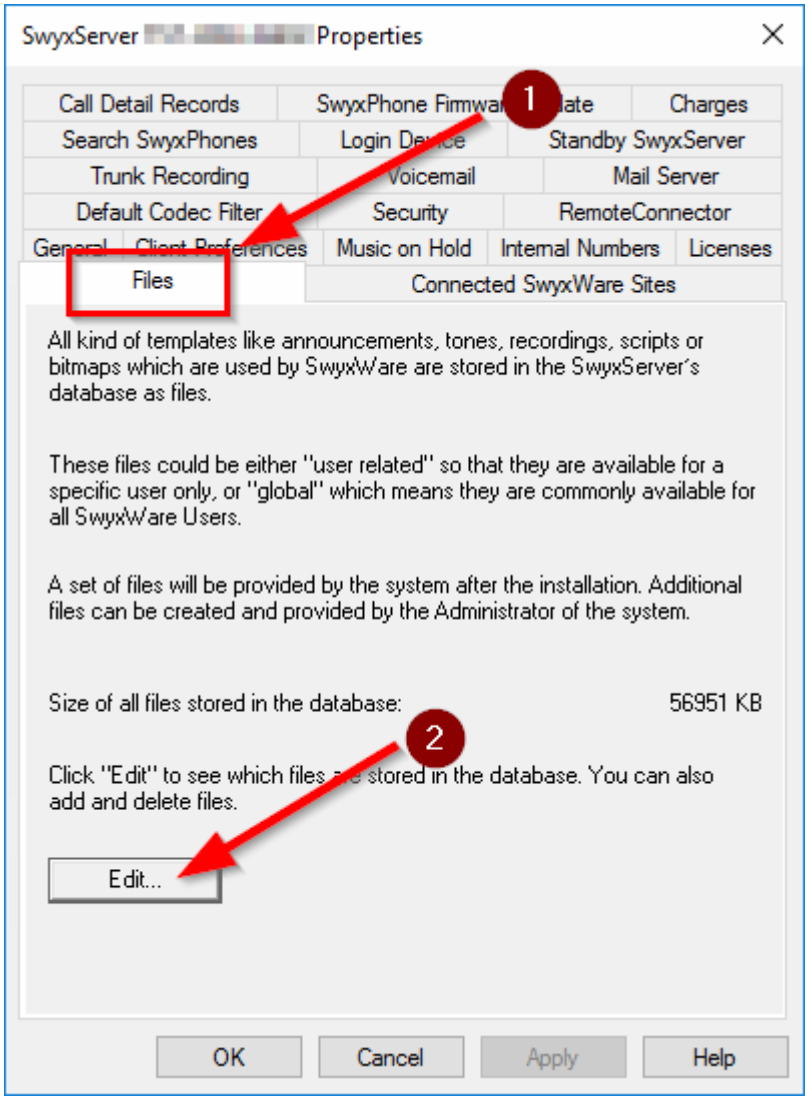
https://www.connectionstrings.com/

Setup

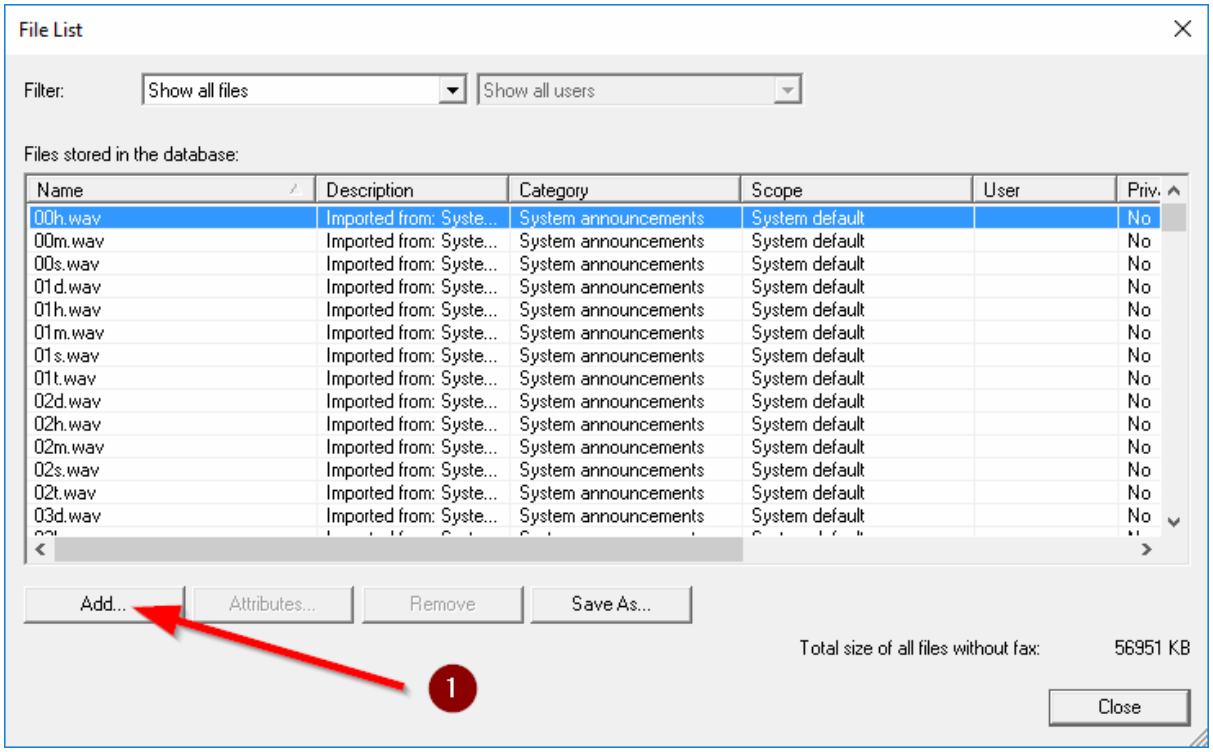
Step 1 - Extract the downloaded .zip file to your local hard drive. Make sure to keep its included folder structure.

Step 2 - Open the SwyxWare Administration and open the **properties** of your Swyx Server.

Step 3 - Switch to the **Files** page and click on **Edit...**



Step 4 - Click on **Add...**

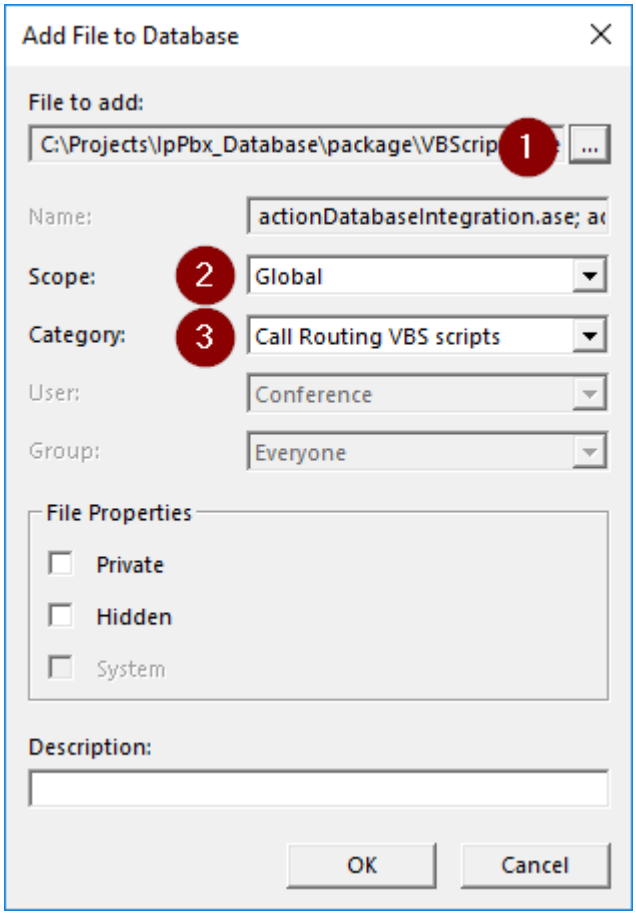


Step 5

1. Select all files from the **ase** folder from the download package.

2. Select **Global** as Scope. (you can also select **User** to load the files into the user scope a your script user or **Group** to load the files into the group scope of your script group)

3. Select **CallRoutingScripts** (in SwyxWare < 13.10) or **Call Routing VBS scripts** (in SwyxWare >= 13.10) as **Category**.



The Database Integration is now installed and can be used within your call routing scripts.

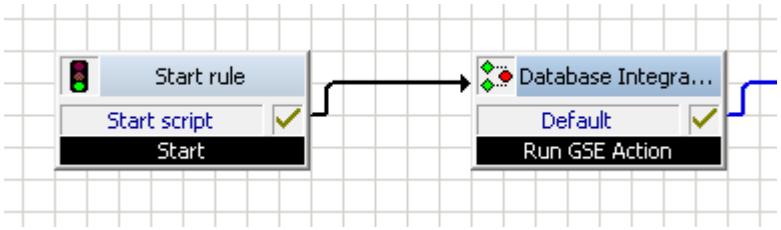
Usage

How in include it into your own call routing

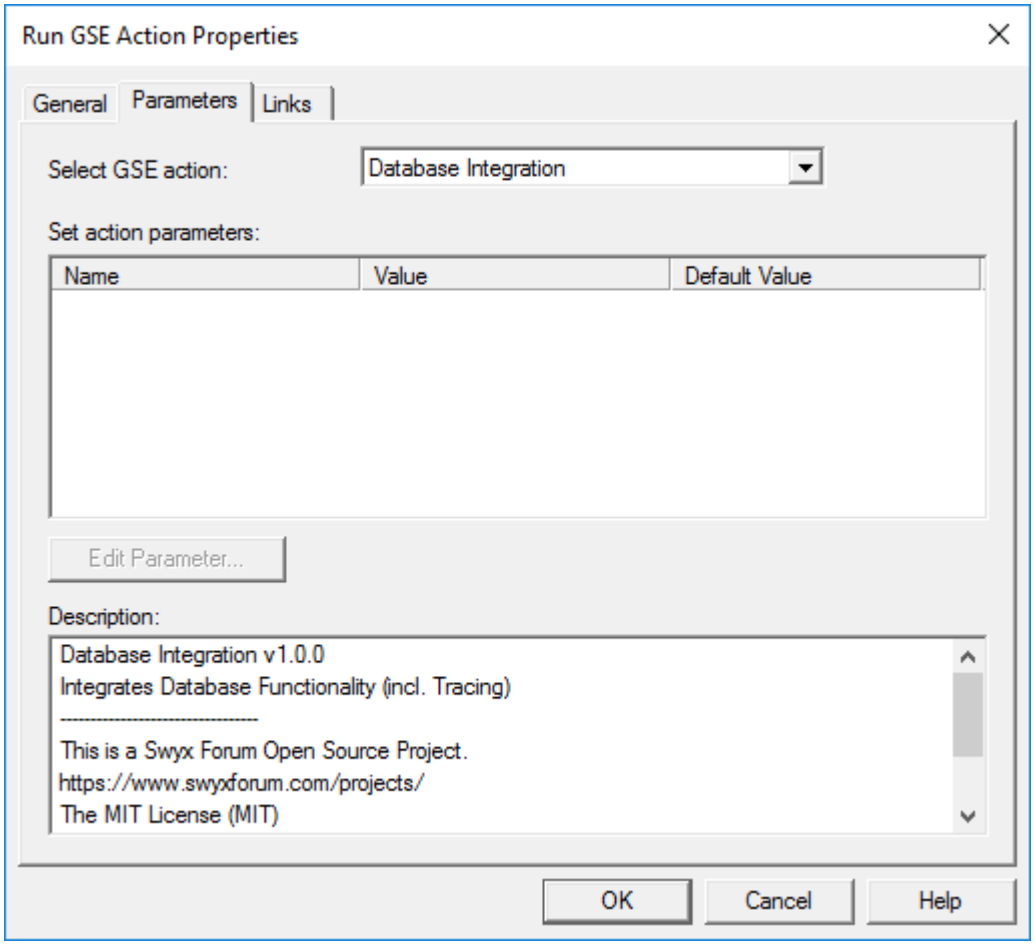
There are two ways to integrate the “Database Integration” into your call routing.

Via GSE Action

Within a new GSE script, insert a **Run GSE Action** block. Best is to locate it right behind the **Start** block.



Double click the new block, switch to the **Properties** page and select the **Database Integration** action.



The code for the Database Integration is now part of your call routing and can be used.

Via include Statement

You can also include the Database Integration directly from within your VBScript code, using an **#include** statement:

```
'#include "IpPbx_Database.vbs"
```

This is by the way exactly (and the only thing) what the **Database Integration** GSE Action is doing in its **Start** block.

The code for the Database Integration is now part of your call routing and can be used.

Read Data

The following code demonstrates how to

- **instantiate** an **IpPbx_Database** object
- **open** a connection to the database
- **create** a new query
- **open** (i.e. run) a query in “read only” mode
- write some of the returned data into the server trace
- **close** and **release** the **IpPbx_Database** object and its included query object (**IpPbx_Query**)

```
Dim db
Set db = New IpPbx_Database

If db.Open _
    ("Provider=sqloledb;" & _
    "Data Source=POLHEIM\SQLEXPRESS;" & _
    "Initial Catalog=IpPbxExtensions;" & _
    "Integrated Security=SSPI") Then

    Dim q
    q = db.NewQuery

    If db.Query(q).OpenReadOnlyForwardOnly _
        ("select * from PersistentVariables") Then

        While Not db.Query(q).bEOF

            PBXScript.OutputTrace "Name = " & _
                db.Query(q).Fields("Name")

            db.Query(q).MoveNext

        Wend

    End If

End If

Set db = Nothing
```

This code is taken from the **Example 2_OpenReadOnlyForwardOnly.vbs** file from the **vbs** folder of the installation package.

See the **Properties and Methods** chapter for details of all the used properties and methods.

Edit Data

The following code demonstrates how to

- **instantiate** an **IpPbx_Database** object
- **open** a connection to the database
- **create** a new query
- **open** (i.e. run) a query in “edit” mode
- **edit** some data in the first dataset of the returned query
- **update** the changed data in the database
- **close** and **release** the **IpPbx_Database** object and its included query object (**IpPbx_Query**)

```
Dim db
Set db = New IpPbx_Database

If db.Open _
    ("Provider=sqloledb;" & _
    "Data Source=POLHEIM\SQLEXPRESS;" & _
    "Initial Catalog=IpPbxExtensions;" & _
    "Integrated Security=SSPI") Then

    Dim q
    q = db.NewQuery

    If db.Query(q).OpenEdit _
        ("select * from PersistentVariables") Then

        If Not db.Query(q).bEOF Then

            db.Query(q).Fields("Name") = "Hello"
            db.Query(q).Update

        End if

    End If

End If

Set db = Nothing
```

This code is taken from the **Example 2_OpenEdit.vbs** file from the **vbs** folder of the installation package.

See the **Properties and Methods** chapter for details of all the used properties and methods.

Add Data

The following code demonstrates how to

- **instantiate** an **IpPbx_Database** object
- **open** a connection to the database
- **create** a new query
- **open** (i.e. run) a query in “add” mode
- **set** some data in a new and empty dataset
- **update** the added data in the database
- **close** and **release** the **IpPbx_Database** object and its included query object (**IpPbx_Query**)

```
Dim db
Set db = New IpPbx_Database

If db.Open _
    ("Provider=sqloledb;" & _
    "Data Source=POLHEIM\SQLEXPRESS;" & _
    "Initial Catalog=IpPbxExtensions;" & _
    "Integrated Security=SSPI") Then

    Dim q
    q = db.NewQuery

    If db.Query(q).OpenAdd _
        ("PersistentVariables") Then

        db.Query(q).Fields("Name")  = "NewName"
        db.Query(q).Fields("Value") = "NewValue"
        db.Query(q).Fields("Scope") = 1

        db.Query(q).Update

    End If

End IF

Set db = Nothing
```

This code is taken from the **Example 2_OpenAdd.vbs** file from the **vbs** folder of the installation package.

See the **Properties and Methods** chapter for details of all the used properties and methods.

Properties & Methods

IpPbx_Database

Property sVersion

This property returns a **string** value and is **read only**.

The returned value is the current version of the installed IpPbx_Database object.

Property nState

This property returns a **numeric** value and is **read only**.

The returned value is the current state of the database connection. There are handy enums defined for each possible value, which are:

- adStateClosed (0)
- adStateOpen (1)
- adStateConnecting (2)
- adStateExecuting (4)
- adStateFetching (8)

Property sState

This property returns a **string** value and is **ready only**.

The returned value is a name of the defined enum of the current state of the database connection. So for a closed connection this property returns “adStateClosed” while nState returns 0.

Property nCommandTimeout

This property sets or returns a numeric value.

The value is the so called CommandTimeout (in seconds) of the database connection. The IpPbx_Database object sets this value to 600 (10 Minutes) by default.

Property nLastError

This property returns a **numeric** value and is **ready only**.

The value is the error code of the last executed command (method). It is 0 if everything went fine, or another number in case of an error.

All occurring errors (code and text) within the IpPbx_Databsse object will automatically be traced into the server trace file.

This is a list of all general Windows error codes:
https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes

This is a list of all database (ADO) related error codes:
https://learn.microsoft.com/en-us/office/client-developer/access/desktop-database-reference/ado-error-codes

In general it has to be said, that the IpPbx_Database object does all error handling (including tracing) for you, so there is no direct need to check this property.

Property sLastErrorDescr

This property returns a **string** value and is **read only**.
The value is the error text the component in which the error occurred returns.

All occurring errors (code and text) within the IpPbx_Databsse object will automatically be traced into the server trace file.

In general it has to be said, that the IpPbx_Database object does all error handling (including tracing) for you, so there is no direct need to check this property.

Method Open

This method **opens** a connection to a database.

It returns a **boolean** value of **True** if the connection was opened successfully or **False** in case of a problem.

To specify the database connection parameters it takes a so called connection string a parameter.

Parameter:

sDSN	string	The connect string defining the database connection details Examples can be found at: https://www.connectionstrings.com/

Returns:

	boolean	True – connection opened, False – some kind of error

Method Close

This method closes a connection to a database.

It returns a **boolean** value of **True** if the connection was closed successfully or **False** in case of a problem.

You don't necessarily need to call close yourself at the end of the call routing script as the IpPbx_Database object will do this automatically for you.

Returns:

	boolean	True – connection close, False – some kind of error

Method NewQuery

This method creates a new query and returns a handle to it. The handle is needed afterwards to access this query. It is possible to create any number of new queries, with all having different handles.

It returns a **numeric** value, which is the handle to the newly created query.

The handle of a query becomes invalid after the DisposeQuery method was called for it.

Returns:

	integer	The handle to the newly created query.

Method ValidHandle

This method checks if the given handle points to a currently valid query. A handle is valid if it points to an existing query, regardless if the query is opened or close. A handle becomes invalid after the DisposeQuery method was called for it.

It returns a **boolean** value of **True** if the handle is valid or **False** if the handle is not valid.

Parameter:

nHandle	integer	handle to check

Returns:

	boolean	True – valid handle, False – invalid handle

Method DisposeQuery

This message disposes (deletes) a query. With that its handle becomes invalid and can't be used anymore.

It returns a **boolean** value of **True** if the query was disposed or **False** in case of a problem.

You don't necessarily need to call DisposeQuery yourself at the end of the call routing script as the IpPbx_Database object will do this automatically for you.

Parameter:

nHandle	integer	handle to query that should get disposed

Returns:

	boolean	True – query disposed, False – some kind of error

IpPbx_Query

Property sVersion

This property returns a **string** value and is **read only**.

The returned value is the current version of the installed IpPbx_Query object.

Property nState

This property returns a **numeric** value and is **read only**.

The returned value is the current state of the query. There are handy enums defined for each possible value, which are:

- adStateClosed (0)
- adStateOpen (1)
- adStateConnecting (2)
- adStateExecuting (4)
- adStateFetching (8)

Property sState

This property returns a **string** value and is **ready only**.

The returned value is a name of the defined enum of the current state of the query. So for a closed query this property returns “adStateClosed” while nState returns 0.

Property bEOF

This property returns a **boolean** value and is **read only**.

This flag indicates if the cursor (pointer to a dataset) in the current recordset (query result) points **behind** the last dataset. When opening a query that returns no data at all, this flag is immediately set. Otherwise it will be set by the “move” methods.

When trying to access data in a query (read or write) if this flag is set, an error occurs (as the cursor points to invalid data). The IpPbx_Query object however provides proper error handling in case you ignore this flag in your code.

Property bBOF

This property returns a **boolean** value and is **read only**.

This flag indicates if the cursor (pointer to a dataset) in the current recordset (query result) points **before** the first dataset. When opening a query that returns no data at all, this flag is immediately set. Otherwise it will be set by the “move” methods.

When trying to access data in a query (read or write) if this flag is set, an error occurs (as the cursor points to invalid data). The IpPbx_Query object however provides proper error handling in case you ignore this flag in your code.

Property Fields

This property sets or returns the fields of a dataset of the current query. The data type depends on the database field which is accessed.

The fields of a dataset can be accessed by their **index** (numerical value starting at 0 in the order the fields are request in the query) or by the field **name**.

```
' by index

Dim sName
sName = db.Query(q).Fields(3)


' by name

db.Query(q).Fields("Name") = "Tom"
```

In order to be able to write into a field, the query must be opened either in "**edit**" or "**add**" mode. This can be done by the methods

- OpenAdd
- OpenEdit

or by the method

- Open

with the cursor types adOpenDynamic, adOpenKeyset or adOpenStatic and the lock type unequal to adLockReadOnly.

To read a value of a field it is enough to open the query in "**read only**" mode.
This can be done with the methods:

- OpenReadOnly
- OpenReadOnlyForwardOnly

or by the method

- Open

with the cursor type adOpenForwardOnly and the lock type adLockReadOnly.

Property nLastError

This property returns a **numeric** value and is **ready only**.

The value is the error code of the last executed command (method). It is 0 if everything went fine, or another number in case of an error.

All occurring errors (code and text) within the IpPbx_Query object will automatically be traced into the server trace file.

This is a list of all general Windows error codes:
https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes

This is a list of all database (ADO) related error codes:
https://learn.microsoft.com/en-us/office/client-developer/access/desktop-database-reference/ado-error-codes

In general it has to be said, that the IpPbx_Query object does all error handling (including tracing) for you, so there is no direct need to check this property.

Property sLastErrorDescr

This property returns a **string** value and is **read only**.
The value is the error text the component in which the error occurred returns.

All occurring errors (code and text) within the IpPbx_Databsse object will automatically be traced into the server trace file.

In general it has to be said, that the IpPbx_Query object does all error handling (including tracing) for you, so there is no direct need to check this property.

Method Open

This function opens a query.

It returns a **boolean** value of **True** if the query was opened or **False** in case of a problem.

Opening a query only works with a closed query. So, if you have an already open query from before either close it first with the Close method or create a new query with the NewQuery method of the IpPbx_Database object.

The IpPbx_Query object uses the ADODB Recordset Open method, so it uses also the same parameters (beside the "ActiveConnection" parameter, which is not needed here).

Parameter:

sSQL	string	The sql statement for the query or name of a table you want to add a new dataset into.
nCursorType	integer	The cursor type. Valid values can be taken from here: https://learn.microsoft.com/en-us/office/client-developer/access/desktop-database-reference/cursortypeenum
nLockType	integer	The lock type. Valid values can be taken from here: https://learn.microsoft.com/en-us/office/client-developer/access/desktop-database-reference/locktypeenum
nCommandType	integer	The command type. Valid values can be taken from here: https://learn.microsoft.com/en-us/office/client-developer/access/desktop-database-reference/commandtypeenum

Returns:

boolean	True – query opened, False – some kind of error

In case you don't want to bother with all those nCursorType, nLockType and nCommandType parameters, the IpPbx_Query object provides some more easy to use Open methods, which practically only ask the sql statement or table name from you:

- OpenReadOnly
- OpenReadOnlyForwardOnly
- OpenEdit
- OpenAdd

Method OpenReadOnly

This method opens the query in "**read only**" mode. This means you can't edit any queried data, i.e. can't call any of the Update methods on it.

It returns a **boolean** value of **True** if the query was opened or **False** in case of a problem.

Once the query is opened you can **move** the cursor, the pointer to the current dataset within the query, **freely** up and down using any of the Move methods.

Opening a query only works with a closed query. So, if you have an already open query from before either close it first with the Close method or create a new query with the NewQuery method of the IpPbx_Database object.

Parameter:

sSQL	string	The sql statement for the query.

Returns:

	boolean	True – query opened, False – some kind of error

Method OpenReadOnlyForwardOnly

This method opens the query in "read only" mode. This means you can't edit any queried data, i.e. can't call any of the Update methods on it.

It returns a **boolean** value of **True** if the query was opened or **False** in case of a problem.

Once the query is opened you can **move** the cursor, the pointer to the current dataset within the query, **only downward** using the MoveNext or MoveLast methods.

Opening a query only works with a closed query. So, if you have an already open query from before either close it first with the Close method or create a new query with the NewQuery method of the IpPbx_Database object.

Parameter:

sSQL	string	The sql statement for the query.

Returns:

	boolean	True – query opened, False – some kind of error

Method OpenEdit

This method opens the query in “edit” mode. It is supposed to be used to edit the data with **one dataset** of the query and then call the **Update** method to make the change permanent in the database.

It returns a **boolean** value of **True** if the query was opened or **False** in case of a problem.

Once the query is opened you can **move** the cursor, the pointer to the current dataset within the query, **freely** up and down using any of the Move methods.

Opening a query only works with a closed query. So, if you have an already open query from before either close it first with the Close method or create a new query with the NewQuery method of the IpPbx_Database object.

Parameter:

sSQL	string	The sql statement for the query.

Returns:

	boolean	True – query opened, False – some kind of error

Method OpenEditBatch

This method opens the query in “edit” mode. It is supposed to be used to edit **multiple datasets** of the query and the call the **UpdateBatch** method to make all changes permanent in the database.

It returns a **boolean** value of **True** if the query was opened or **False** in case of a problem.

Once the query is opened you can **move** the cursor, the pointer to the current dataset within the query, **freely** up and down using any of the Move methods.

Opening a query only works with a closed query. So, if you have an already open query from before either close it first with the Close method or create a new query with the NewQuery method of the IpPbx_Database object.

Parameter:

sSQL	string	The sql statement for the query.

Returns:

	boolean	True – query opened, False – some kind of error

Method OpenAdd

This method opens the query in “add” mode. This means you can edit all fields of the new dataset for the given table.

Note: after setting the fields you need to call the **Update** method to make your changes permanent within the database!

Opening a query only works with a closed query. So, if you have an already open query from before either close it first with the Close method or create a new query with the NewQuery method of the IpPbx_Database object.

Parameter:

sSQL	string	Name of the table you want to add a new dataset to.

Returns:

	boolean	True – query opened, False – some kind of error

Method Close

This method closes a query after you have opened it before. After you have closed the query you can open it again for another query against the database.

It returns a **boolean** value of **True** if the query was closed or **False** in case of a problem.

You don’t need to close the query yourself at the end of your call routing script. The IpPbx_Query object does that for you automatically.

Returns:

	boolean	True – query closed, False – some kind of error

Method Dump

This method dumps the entire content of the query (field names and data) after you opened it into the server trace file.

This method is meant for testing or debugging purposes.

This method only works if you open the query with a freely movable cursor. This is either the case with using the

- OpenReadOnly
- OpenEdit

methods, or the

- Open

method with the cursor types adOpenDynamic, adOpenKeyset or adOpenStatic.

This method doesn't have any parameters or return values.

Method MoveFirst

This method moves the cursor, i.e. the pointer to the current dataset within the opened query, to the first position, that is the first dataset in the query.

It returns a **boolean** value of **True** if the cursor was moved or **False** in case of a problem.

This method only works if you open the query with a freely movable cursor. This is either the case with using the

- OpenReadOnly
- OpenEdit

methods, or the

- Open

method with the cursor types adOpenDynamic, adOpenKeyset or adOpenStatic.

Note: you must not use the MovePrevious method afterwards to move the cursor any further up.

Returns:

boolean	True – cursor moved, False – some kind of error

Method MovePrevious

This method moves the cursor, i.e. the pointer to the current dataset within the opened query, to the previous position, that is the previous dataset to the current position in the query.

It returns a **boolean** value of **True** if the cursor was moved or **False** in case of a problem.

This method only works if you open the query with a freely movable cursor. This is either the case with using the

- OpenReadOnly
- OpenEdit

methods, or the

- Open

method with the cursor types adOpenDynamic, adOpenKeyset or adOpenStatic.

Note: you need to check the bBOF property if after you moved the cursor it still points to a valid dataset. The bBOF property is True if you moved the cursor before the first dataset.

Returns:

boolean	True – cursor moved, False – some kind of error

Method MoveNext

This method moves the cursor, i.e. the pointer to the current dataset within the opened query, to the next position, that is the next dataset to the current position in the query.

It returns a **boolean** value of **True** if the cursor was moved or **False** in case of a problem.

This method only works if you open the query with a freely movable cursor. This is either the case with using the

- OpenReadOnly
- OpenEdit

methods, or the

- Open

method with the cursor types adOpenDynamic, adOpenKeyset or adOpenStatic.

Note: you need to check the bEOF property if after you moved the cursor it still points to a valid dataset. The bEOF property is True if you moved the cursor after the last dataset.

Returns:

boolean	True – cursor moved, False – some kind of error

Method MoveLast

This method moves the cursor, i.e. the pointer to the current dataset within the opened query, to the last position, that is the last dataset in the query.

It returns a **boolean** value of **True** if the cursor was moved or **False** in case of a problem.

This method only works if you open the query with a freely movable cursor. This is either the case with using the

- OpenReadOnly
- OpenEdit

methods, or the

- Open

method with the cursor types adOpenDynamic, adOpenKeyset or adOpenStatic.

Note: you must not use the MoveNext method afterwards to move the cursor any further down.

Returns:

boolean	True – cursor moved, False – some kind of error

Method Update

This method updates (i.e. stores) the current dataset of the query (i.e. the dataset the cursor points to) into the database. It can not be used to update (i.e. store) multiple datasets at once.

This function only make sense to get called if you have opened the query in either “edit” or “add” mode. This can be done using the

- OpenAdd
- OpenEdit

methods, or the

- Open

method with the cursor types adOpenKeyset and or adOpenKeyset.

If you modify multiple datasets within your query, you need to call Update for every dataset separately. Or you use **OpenEditBatch**/**UpdateBatch**.

Returns:

boolean	True – query updated, False – some kind of error

Method UpdateBatch

This method updates (i.e. stores) the current query into the database. If you made any modifications in it beforehand this function finally makes your changes permanent.

This function only make sense to get called if you have opened the query in either “edit” or “add” mode. This can be done using the

- OpenEditBatch

method or the

- Open

method with adOpenKeyset and adLockBatchOptimistic.

If you only need to modify only one dataset within your query, you use can also use **OpenEdit**/**Update**.

Returns:

boolean	True – query updated, False – some kind of error

Examples

Example 1_NewQuery.vbs

This example shows how to instantiate an IpPbx_Database object and create a new query.

Example 2_Open.vbs

This example shows how to open (i.e. run) a query. This is a general version of opening a query. For the ease of use there are more specialised “Open” methods available, which require less parameters and are shown in the following examples.

Example 2_OpenAdd.vbs

This example shows how to open a query in “add” mode, add new data into a new dataset and update (store) it into the database.

Example 2_OpenEdit.vbs

This example shows how to open a query in “edit” mode, edit some data in **one** dataset of the queried data and update (store) it into the database.

Example 2_OpenEditBatch.vbs

This example shows how to open a query in “edit” mode, edit data in **multiple** datasets in the queried data and update (store) all changes into the database.

Example 2_OpenReadOnly.vbs

This example shows how to open a query in “read only” mode and dump all queried data into the server trace file.

Example 2_OpenReadOnlyForwardOnly.vbs

This example shows how to open a query in “read only” mode and navigate through all queried data. The difference to the “OpenReadOnly” method is that is possible to move forward (down) through the queried data only. This comes with a huge performance advantage.

Example 3_Dump.vbs

This example shows how to query data from a database and dump all of it into the server trace file.

Example 4_Field.vbs

This example shows how to query data from a database and access a specific field of the queried data.

Example 5_MoveNext.vbs

This example shows how to query data from a database and run through all queried data using the “MoveNext” method to go from one dataset to the next one.

Example 6_Update.vbs

This example shows how to query data from a database, edit a specific field within **one dataset** of the queried data and update (store) it into the database.

Example 6_UpdateBatch.vbs

This example shows how to query data from a database, edit a specific field within **multiple datasets** within the queried data and update (store) everything afterwards into the database.