RemoteAPI Guide

Contacting DevTech Services, LLC

DevTech Services, LLC
PO Box 1153
Royse City, Texas 75189

Office 469-585-1204
Fax 914-992-0255
info@devtechservices.com
http://www.devtechservices.com

Contents

Installation

Web Service

Download and install .Net Framework 3.5 SP1 if needed.

1. Copy the WebService directory from the install media to a suitable location on the SDE server.
2. Create an appropriate virtual directory to the WebService directory.
    a. Ensure that the ASP.Net level on the virtual directory is set to .Net 2.
    b. Ensure that the virtual directory is using an appropriate application pool.
3. Create a new local directory where the RemoteAPI webservice can save temporary uploaded files. The ASPNET local user should have full control of this folder.
4. Modify the web.config file in the WebService directory.
    a. MagicUser = user through whom access to the SDE database will be handled
    b. MagicPwd = the password of the user
    c. MagicGroup = the name of a group to which the MagicUser belongs
    d. DSN = the ODBC data source name used to connect to the SDE database.
    e. FileSaveToLocation = the location to which uploaded attachments will be temporarily stored. The ASPNET user must have full control of this location.

The web service will now be available as a web reference via its URL:

http://<server_name>/<virtual directory>/DTRemoteAPI.asmx

DTAPI server libraries

Copy RemoteAPI folder as needed. Reference the DTAPIHandler.dll as needed during application development.

RemoteAPI Client

Copy RemoteAPI folder as needed. Reference the RemoteAPI.dll as needed during application development.

Structure

DTAPIHandler.Handler class definition

This library defines methods for interacting with the SDE database.   This library is installed on an SDE web application server. This library is the core component of the RemoteAPI. It handles connecting to the SDE database and all data operations. The majority of data operations are handled through the Handler class. The web service and RemoteAPI client depend on the XMLHandler method of the Handler class.

Method ***Connect***(UID As String, PWD As String, Group As String, DSN As String, Optional LoggingLevel As LogLevel) - returns void; this is the first method that must be called after instantiation the oHandler class. The UID, PWD, Group, and DSN must all correspond to available settings on the SDE application server. The user specified by UID must be a member of the Group provided.

Instantiation: DTAPIHandler.mHandler

Method ***SetUserConnection***(mUserName As String, mPwd As String, GroupName As String) - returns null; sets the connection credentials used by the RemoteAPI when interacting with SDE. The user connection is initially set using the credentials supplied to the Connect() method. This method will override those credentials without re-initializing the component.

Property ***View*** - string; gets or sets the SDE view that will be accessed. This must be set before any operations are defined.

```
View = “Incident”
```

Method ***SetField***(Name As string, Value As object) - returns void; sets the value of the field identified by “Name” in the current SDE View. Use this method to set the specific data values on fields that will then be inserted or updated in the SDE database.

```
SetField(“Subject ID”, “HARDWARE”)
```

Note: When setting values for virtual fields, the foreign key link from the specified module will not be populated unless a value virtual represented a unique field in the foreign module is specified. For example, to set the client on an Incident, use the Client ID field since it is a unique field in the Clients module.

Property ***MD*** - object; gets the SDE Metadata object loaded during initialization.

Method ***Insert*** - returns void; inserts a new record into the SDE database, as directed by the SetField and View objects.

Method ***Update***(optional sFilter as string) - returns void; when sFilter is supplied, the supplied "SetField" values are pushed to the database on the records matching the SQL in sFilter in the view specified by the View property. When sFilter is not supplied, the handler attempts to build a SQL WHERE clause using any unique fields and values supplied by the SetField method.

Method ***Delete***(optional sFilter as string) - returns void; if sFilter is supplied, SDE records matching the SQL filter supplied within the view specified in the "View" property are deleted. If no filter is supplied, the handler attempts to create a filter from supplied "SetField" field names and values.

Method ***Retrieve***(strFilter as string) returns void; retrieves the records in the View that meet the SQL WHERE filter in strFilter. The resulting records are stored in the Records collection.

Method ***HandleXMLData***(XMLData as string) - XMLData is an XML formatted string containing elements specifying how to interact with SDE. Returns a string in XML format derived from the specifications in XMLData.

Property ***Records*** - collection; gets the collection of Record objects held in the current instance of the class.

Property ***RecordCount*** - gets integer; returns the count of the records currently in the class instance. Records can be accumulated during create, **retrieve**, and update operations.

Property ***FieldCount*** - gets integer; returns the count of fields in each record currently in the class instance.

Method ***GetField***(RecordIndex As integer, Field As string) - returns string; returns the value of the field in the specified record.

Method ***FieldName***(Index as integer) - returns string; returns the name of field within a returned Record object as located at the position denoted by “Index.”

Method ***Clear*** - returns void; resets the Records collection, the View property, and re-initializes the handler in preparation for a new operation.

Web Service

The RemoteAPI web exposes methods for use by HTTP consumers that are used to interact with SDE. The web service is hosted on an SDE web server.   The sole exposed class in the web service is XMLHandler. XMLHandler can accept properly formatted XML data to define interactions with SDE, and can upload attachments to the SDE database.

The RemoteAPI Client is provided to simplify access to the web service without the need to directly access the methods of the web service.

Method ***AttachFileFromURL***(sURL as string, sSDEModule as string, sSequence as string, sAttachmentsFieldName as string) - returns string (empty if successful.)

sURL – the URL from which the file can be downloaded.

sSDEModule – the name of the SDE module the attachment record will be linked to.

sSequence – the sequence of the record in sSDEModule the attachment will be linked to.

sAttachmentsFieldName – optional. If this is not supplied, XMLHandler finds the first foreign key in the SDE “Attachments” module that leads to the module specified by sSDEModule. This is primarily used for cases where there are multiple foreign keys from “Attachments” to a single SDE module.

Method ***AttachUploadedFile***(strFileName As String, objFile As Byte array, sSDEModule As String, sSequence As Integer, sAttachmentsFieldName As String) – returns string (empty if successful.)

strFileName – the name of the file that is being uploaded to the server.

objFile – a bytearray representing a binary stream of the files contents as sent via an HTTP POST operation.

sSDEModule – the name of the SDE module the attachment record will be linked to.

sSequence – the sequence of the record in sSDEModule the attachment will be linked to.

sAttachmentsFieldName – optional. If this is not supplied, XMLHandler finds the first foreign key in the SDE “Attachments” module that leads to the module specified by sSDEModule. This is primarily used for cases where there are multiple foreign keys from “Attachments” to a single SDE module.

Method ***PostToRemoteAPI*** (sXML As String)– returns string. Accepts formatted XML defining data and action to the SDE server, and the format and data to be returned from the operation.

RemoteAPI Client

The RemoteAPI Client library provides methods for rapidly developing applications that can interact remotely with SDE. The library is accessible in IDE’s that can access Microsoft .Net assemblies, and can be registered for COM operations using the .Net frameworks “regasm” utility.

Instantiation: ***RemoteAPIClient.Client***

Properties and Methods

Method ***New***(optional WebServiceURL as string) – instantiates the class in memory. If WebServiceURL is not provided, it must be set with the WebServiceURL property before the client can communicate with the web service.

Property ***WebServiceURL*** – string; sets the location of the RemoteAPI web service.

Method ***Clear***() – clears all properties previously set in preparation for beginning a new operation.

Property ***SDEModule*** – gets or sets the name of the SDE module to be accessed.

Property ***Action*** – gets or sets the action to be specified when posting to the web site using a value from the DataAction enumeration.

DataAction.create = 0
DataAction. Update =1
DataAction.retrieve = 2
DataAction.delete = 3

Property ***Filter*** – gets or sets a string that defines the SQL WHERE clause to be used in Update, Retrieve, and Delete operations.

Method ***SetSendField***(FieldName as string, FieldValue as string) – sets the value of a field identified by FieldName in the specified SDEModule. Defines data to be send to SDE.

Method ***SetReturnField***(MagicFieldName as string, ReturnFieldName as string) – Used to define the XML returned from an operation. When sending record data back, the web service will retrieve the values for each MagicFieldName for all calls to SetReturnField, and will create an XML "field" element with attribute "name" set to the value of ReturnFieldName. The element's "InnerText" property will be the value from the SDE record for the "MagicFIeldName" field.

Property ***XMLText*** – gets a string built using supplied settings of properties.

Method ***PostToSDE***(optional sXML as string) – returns string; if sXML is provided, the value is sent to the web service. If not, the client builds the XML using values supplied in the SDEModule, SetSendField, SetReturnField, Action, and Filter methods. The web server's XML response is returned.

Property ***RecordCount*** – returns integer; returns the count of records retrieved from SDE

Property ***Records*** – returns a collection of Record objects retrieved from SDE.

Property ***Record***(Index as integer) – returns the Record object specified by the location Index in the Records collection.

Method ***UploadLocalFile***(sFilename as string, sSDEModule as string, iSequence as integer,optional sAttachmentFieldName as string) – uploads a file from the local machine to SDE, and attaches it to a record specified by "iSequence" in the SDE module specified by "sSDEModule." If sAttachmentFieldName is not supplied, the web service will attempt to locate the first foreign key field in the Attachments module that links to the module specified by sSDEModule.

Note: The file specified by sFileName must be accessible by the user under whose credentials the application hosting the RemoteAPI Client is running.

Method ***UploadURLFile***(sURL as string, sSDEModule as string, iSequence as integer, optional sAttachmentFileName as string) – instructs the web service to download a file from the location specified by sURL. If the download is successful, the web service will attach it to a record specified by “iSequence” in the SDE module specified by “sSDEModule.”

Accessing Data returned by an operation

Data returned to the RemoteAPI Client is placed in the RemoteAPI Client’s Records collection. Each record consists of a collection of FieldStructs that contain the field’s Name (as specified in the SetReturnField() method calls,) and the Value of the field corresponding to the SDE record.

A single record can also be accessed by using the Record(Index) property, where Index is an integer from 1 to the RemoteAPI Client’s RecordCount (or Records.Count.)

Example Transaction

This example illustrates using the RemoteAPI Client to create a new Incident record in SDE:

```
        'Instantiate the RemoteAPI Client and direct it to the web service url
        'URL may vary.
        Dim RClient As New RemoteAPIClient.Client(“http://localhost/WSRemoteAPI/DTRemoteAPI.asmx”)

        'Prepare a string to hold our return values from the web service
        Dim sRet As String = ““
        'Set up the data and action to be sent to SDE
        With RClient
            .Action = RemoteAPIClient.Client.DataAction.create
            .SDEModule = “Incident”
            'Set values for the Incident that will be created
            .SetSendField(“Client ID”, “BMORGAN”)
            .SetSendField(“Group Name”, “HELPDESK”)
            .SetSendField(“Login ID Assigned To”, “RSMITH”)
            .SetSendField(“Subject ID”, “HARDWARE”)
            .SetSendField(“Incident Description”, “Having problems with printer.”)
            'Tell the web services that we'd like to have the new Incident # sent back to us.
            .SetReturnField(“Incident #”, “TicketNumber”)
            'Send it over; sRet will contain the raw XML the web service responds with.
            sRet = RClient.PostToSDE()
        End With
        Dim i As Integer = 0
        'RClient.Records is a collection of RemoteAPIClient.Records send back from the web service
        'We'll iterate through each Record and build a string to display the data
        For Each oRecord As RemoteAPIClient.Record In RClient.Records
            i += 1
            sRet += "Record " & i & vbCrLf
            'Here, we'll iterate through the fields of this record.
            'The number of fields and the name of each will be the
            'same from record to record, but the values will vary.
            'Each field is a RemoteAPIClient.FieldStruct object,
            'which means that it has a Name and Value for easy parsing.
            For Each item As RemoteAPIClient.FieldStruct In oRecord.Fields
                sRet += vbTab & "Field " & item.Name & " = '" & item.Value & "'" & vbCrLf
            Next
        Next
        txtResult.Text = sRet
```

After executing this code in a test Windows form, the txtResult text box displays:

```
Record 1
        Field TicketNumber = '31'
```

The raw XML returned by the server is:

```
<records><record><field name="TicketNumber">31</field></record></records>
```

This example shows code to retrieve all of the Incidents assigned to a particular Client.

```
'Instantiate the RemoteAPI Client and direct it to the web service url
'URL may vary.
        Dim RClient As New RemoteAPIClient.Client("http://localhost/wsremoteapi/DTRemoteAPI.asmx")

        'Prepare a string to hold our return values from the web service
        Dim sRet As String = ""
        'Set up the data and action to be sent to SDE
        With RClient
            .Action = RemoteAPIClient.Client.DataAction.retrieve
            .SDEModule = "Incident"
            'Set the filter to apply to the Incident records
            .Filter = " [Client ID] = 'BTEST'"
            'Tell the web services that we'd like to have the new Incident # sent back to us.
            .SetReturnField("Incident #", "TicketNumber")
            .SetReturnField("Group Name", "Group")
            .SetReturnField("Open Date & Time", "Opened On")
            'Send it over; sRet will contain the raw XML the web service responds with.
            sRet = RClient.PostToSDE()
        End With

        Dim i As Integer = 0
        Dim sData As String = ""
        'RClient.Records is a collection of RemoteAPIClient.Records send back from the web service
        'We'll iterate through each Record and build a string to display the data
        For Each oRecord As RemoteAPIClient.Record In RClient.Records
            i += 1
            sData += "Record " & i & vbCrLf
            'Here, we'll iterate through the fields of this record.
            'The number of fields and the name of each will be the
            'same from record to record, but the values will vary.
            'Each field is a RemoteAPIClient.FieldStruct object,
            'which means that it has a Name and Value for easy parsing.
            For Each item As RemoteAPIClient.FieldStruct In oRecord.Fields
                sData += vbTab & "Field " & item.Name & " = '" & item.Value & "'" & vbCrLf
            Next
        Next
        txtResult.Text = sData
```

After executing this code in a test Windows form, the txtResult text box displays:

```
Record 1
        Field TicketNumber = '1'
        Field Group = 'EXTERNAL_SUPPORT'
        Field Opened On = '9/4/2008 12:19:34 PM'
Record 2
        Field TicketNumber = '4'
        Field Group = 'TEST_GROUP'
        Field Opened On = '3/24/2009 1:31:48 AM'
Record 3
        Field TicketNumber = '5'
        Field Group = 'TEST_GROUP'
        Field Opened On = '3/24/2009 1:39:23 AM'
…
```

The raw XML returned by the server is:

```
<records><record><field name="TicketNumber">1</field><field name="Group">EXTERNAL_SUPPORT</field><field name="Opened On">9/4/2008
12:19:34 PM</field></record><record><field name="TicketNumber">4</field><field name="Group">TEST_GROUP</field><field name="Opened
On">3/24/2009 1:31:48 AM</field></record><record><field name="TicketNumber">5</field><field name="Group">TEST_GROUP</field><field
name="Opened On">3/24/2009 1:39:23 AM</field></record></records>
…
```

XML Definitions

Several of the methods of the RemoteAPI can use XML data supplied as an argument. These methods are:

DTAPIHandler.XMLHandler.HandleXMLData(XML string) – this is the core XML handler for the RemoteAPI system.

WSRemoteAPI.PostToRemoteAPI(XML string) – This is the sole method through which the RemoteAPI web service takes direction on how to pass data to and from the SDE database.

RemoteAPI.Client.PostToSDE(optional XML string) – The RemoteAPI client will assemble its own XML string after its other properties are set, but it can also take a manually derived XML string and pass it to the RemoteAPI web service.

Input

Simple Schema

To return a simple record set from Magic, the following XML sample will interface with a single Magic module and return a single recordset.

```
<Magic>
        <module name=“mod_name” action=“create” filter=“SQL WHERE clause”>
                <field name=“Magic field name1”>Value for field</field>
                <field name=“Magic field name2”>Value for field</field>
                <field name=“Magic field name3”>Value for field</field>
                …
        </module>
        <return attrib1=“Value1” …>
                <field name=“Magicfield1”>External field reference1</field>
                <field name=“Magicfield2”>External field reference2</field>
                <field name=“Magicfield3”>External field reference3</field>
                …
        </return>
        …
</Magic>
```

Valid values for the “action” attribute of the “module” element are “create”, “update”, “delete”, and “retrieve.” When “action” is set to “update,” “retrieve”, or “delete,” the “filter” attribute is required.

Any number of attributes can be built into the <return> element. These will be sent back to the client in the <records> header exactly as they are sent to the Remote-API.

Complex Schema
The XML handler is capable of processing different types of requests against multiple Magic modules, and returning different field sets from each. In order to do this, XML containing references to multiple Magic modules can be constructed.

```
<Magic style=“multi”>
        <module name=“mod_name” action=“create” filter=“SQL WHERE clause”>
                <field name=“Magic field name1”>Value for field</field>
                <field name=“Magic field name2”>Value for field</field>
                <field name=“Magic field name3 “>Value for field</field>
                …
                <return attrib1=“Value1” …>
                        <field name=“Magicfield1”>External field reference1</field>
                        <field name=“Magicfield2”>External field reference2</field>
                        <field name=“Magicfield3”>External field reference3</field>
                        …
                </return>
        </module>
        …
</Magic>
```

The only recognized value for the “style” attribute on the <Magic> element is “multi”. This causes the Remote-API to return <records> elements within a top level <data> tag, enhancing the ability to manipulate multiple <records> sets in a well-formatted XML document.

Output

Simple schema
When the “style” attribute is missing from the request’s <Magic> element or its value is anything but “multi”, the Remote-API returns a single <records> element.

```
<records attrib1=“Value1”…>
        <record>
                <field name=“External Field Reference 1”>value from Magic 1</field>
                <field name=“External Field Reference 2”>value from Magic 2</field>
                <field name=“External Field Reference 3”>value from Magic 3</field>
                …
        </record>
        <record>
                <field name=“External Field Reference 1”>value from Magic 1</field>
                <field name=“External Field Reference 2”>value from Magic 2</field>
                <field name=“External Field Reference 3”>value from Magic 3</field>
                …
        </record>
        …
</records>
```

Complex Schema
When the “style” attribute is missing from the request’s <Magic> element or its value is anything but “multi”, the Remote-API returns a top level <data> element populated with one or more <records> element.

Each <records> element corresponds to the <return> element of a single <module> element sent in the request.

```
<data>
        <records attrib1=“Value1”…>
                <record>
                        <field name=“ExternalField1”>value from Magic 1</field>
                        <field name=“ExternalField2”>value from Magic 2</field>
                        <field name=“ExternalField3”>value from Magic 3</field>
                        …
                </record>
                …
        </records>
        …
</data>
```