CRM Web Service API

Overview

The Synthesys CRM Web Services module provides an API into the Synthesys CRM system, which can be used in the following scenarios :

1. Provide an API for 3rd party applications to call into CRM.
2. Provide an API for Synthesys.NET to access CRM.
3. Provide an API for Synthesys v5 to access CRM, to replace current RunnerCOM and direct table access.
4. Provide an API that will provide functionality for web self service portal (username/password access for customers, not users).

API details

Unicode
Currently (Synthesys v4) the Synthesys CRM system is not Unicode capable. Synthesys v5 will support Unicode.

Security
There are currently three ways of authorising access to the Synthesys CRM API :

1. Token based. With this, a call to AuthenticateUser should be used to obtain an XML Authentication String which can be used as input to subsequent API calls.
2. Per-call – username and password are supplied and checked for each API call.
3. Unsecured – all callers of the API are trusted. This mode is not enabled by default, and should only be turned on if access to the web service is controlled by method external to Synthesys (such as IPSEC).

In each case, if the calling user has sufficient privileges, they can then call different API calls, and also perform actions on behalf of other users.

Security API Functions

AuthenticateUser

```
string AuthenticateUser(string Username, string Password)
```

This returns an XML authentication string which can be used as input to subsequent APIs. Note that AuthenticateUser is only for authenticating users of the API; there is an AuthenticateCustomer call for use in web applications which require password checking for customers.

Example call :

```
Token = SynthesysCRM.AuthenticateUser("Luke", "luke");
```

Example return :

```
<Authentication>
  <Token>3e385613-302c-4294-9b31-8ce7650260d8</Token>
  <Username>Luke</Username>
</Authentication>
```

AuthenticationString

```
<Authentication>
<Token>Guid</Token>
<AuthenticatedUser>Username</AuthenticatedUser>
<OnBehalfOf>Username</OnBehalfOf>
```

```
<Username>Username</Username>
<Password>Password</Password>
</Authentication>
```

The **<Token>** element is present if using an authenticating string returned from AuthenticateUser. If using per-call authentication, then the **<Username>** and **<Password>** elements should be present. If unsecured, neither element needs to be present.

We would normally recommend using per-call authentication for ease of use. Logon tokens hold the risk of becoming invalid over time or if the service is restarted; unsecured access can be used in an entirely trusted installation environment.

<OnBehalfOf> is used when performing actions on behalf of a user (this a the trusted subsystem model rather than a delegation model of security). So for example, the user of the system might be 'CCAdministrator', but this user might call the API to add a history item for user 'John'. The component calling the API would not have the password for John so could not call the API using credentials for John, but can call the API using its own credentials, and then has rights to act on behalf of John.

Metadata API Functions

ListPrefixes

```
string ListPrefixes(string AuthenticationString)
```

Returns a list of all the available CRM tennants in the system.

Example return :

```
<Prefixes>
  <Prefix>ELECT</Prefix>
  <Prefix>EXTERN</Prefix>
  <Prefix>LUKE</Prefix>
</Prefixes>
```

GetCustomerDescription

```
string DescribeCustomer(string AuthenticationString, string Prefix)
```

Returns the meta data for a CRM tennant, the fields and attributes of the customer. This can be used to dynamically construct a user interface for the CRM.

Example Call :

```
result = SynthesysCRM.DescribeCustomer(Token, "LUKE");
```

Example return (abbreviated) – see below for more details

```
<CustomerDescription Name="LUKE">
  <Property Id="P001" Name="Customer ID" Type="String"
IsCustomerId="True" />
  <Property Id="P002" Name="Name" Type="String" IsName="True"
Maximum="40" IsIndexed="True" />
... part omitted ...
  <Property Id="P021" Name="Active" Type="Boolean" IsRequired="True"
IsActive="True" />
  <Property Id="P022" Name="Password" Type="String" Maximum="44"
IsPassword="True" />
  <Property Id="P023" Name="Salt" Type="String" Maximum="8"
IsSalt="True" />
</CustomerDescription>
```

GetHistoryTypes

```
string GetHistoryTypes(string AuthenticationString, string Prefix)
```

Returns the different history types associated with a CRM tennant.

Example Call :

```
result = SynthesysCRM.GetHistoryTypes(Token, "LUKE");
```

Example return :

```
<HistoryTypes Prefix="LUKE">
  <HistoryType>CallDeleted</HistoryType>
  <HistoryType>CallQueued</HistoryType>
  <HistoryType>CallRecording</HistoryType>
  <HistoryType>Customer Deleted</HistoryType>
  <HistoryType>Customer Details Altered</HistoryType>
  <HistoryType>Customer Structure Altered</HistoryType>
  <HistoryType>Customer Structure Created</HistoryType>
  <HistoryType>I/B Call</HistoryType>
  <HistoryType>I/B Call Failed</HistoryType>
  <HistoryType>New Customer Created</HistoryType>
  <HistoryType>Note</HistoryType>
  <HistoryType>O/B Call</HistoryType>
  <HistoryType>O/B Call Failed</HistoryType>
  <HistoryType>Predictive Dialler</HistoryType>
  <HistoryType>Product</HistoryType>
  <HistoryType>Report Run</HistoryType>
  <HistoryType>Runner Dial</HistoryType>
  <HistoryType>SQI Queue Result</HistoryType>
</HistoryTypes>
```

Customer API Functions

GetCustomer

```
string GetCustomer(string AuthenticationString, string Prefix, string
CustomerId)
```

Get the data for a single customer. See section below for notes on the **<Customers>** XML document structure.

Example call :

```
result = SynthesysCRM.GetCustomer(Token, "LUKE", "LUKE_1");
```

Example return :

```
<Customers>
  <Customer Prefix="LUKE">
    <Property Name="Customer ID" Value="LUKE_1" />
    <Property Name="Name" Value="Luke Harris" />
    <Property Name="Telephone" Value="" />
    <Property Name="Enumeration" Value="Blue" />
    <Property Name="Field1" Value="" />
    <Property Name="Field2" Value="" />
    <Property Name="Field3" Value="" />
    <Property Name="Line1" Value="" />
    <Property Name="Line2" Value="" />
    <Property Name="Line3" Value="" />
    <Property Name="Line4" Value="" />
    <Property Name="Line5" Value="" />
    <Property Name="Line6" Value="" />
    <Property Name="PostCode" Value="" />
    <Property Name="Pet Name" Value="Hamish" />
```

```
    <Property Name="DateTime" Value="" />
    <Property Name="Number" Value="" />
    <Property Name="BoolType" Value="False" />
    <Property Name="Active" Value="True" />
    <Property Name="Password" Value="*****" />
    <Property Name="Salt" Value="NaCl" />
  </Customer>
</Customers>
```

SearchCustomers

```
string SearchCustomers(string AuthenticationString, string Prefix,
string Search, bool DoWildcardSearch, int MaximumResults)
```

Searches for a customer.

The Search string is a partial customer record; those fields present are the search criteria.

Example calling sequence – this searches for the first ten inactive customers :

```
XElement xelement = new XElement("Customer",
    new XElement("Property",
        new XAttribute("Name", "Active"),
        new XAttribute("Value", "False"))
    );
//XElement xelement = new XElement("Customer");
result = SynthesysCRM.SearchCustomers(Token, "LUKE",
xelement.ToString(), false, 10);
```

Example return is the same as for GetCustomer, but might contain more than one **<Customer>** element.

Notes on searches :

1. By default, deactivated customers are not returned. If you want deactivated customers, then ask for them specifically in the search criteria (as in the example above).
2. By default we will do 'AND' logic on the search criteria.
3. Format is as for a SQL LIKE statement, so % and _ and [] for wildcard characters. Most users are more familiar with * so you might choose to replace asterisks with percent characters before calling this function.
4. Since enumerated types are stored in the database as integers, it isn't possible to do wildcard searches on these (although exact searches are supported).
5. Wildcard searches on datetimes aren't recommended[1].
6. Searches on passwords will always fail to match records. Use AuthenticateCustomer for this functionality.

AuthenticateCustomer

```
string AuthenticateCustomer(string AuthenticationString, string
Prefix, string Search)
```

AuthenticateCustomer will return customer ID if successful, or an empty string if authentication failed. Reasons for failure are not given in the API call for security reasons, but the reason (customer not found or password incorrect) can be found in the server log files.

[1] This is because SQL rules mean that dates are considered to be in the format 'Jan 1, 1900 9:20AM' for example, so getting the right search string can be tricky.

Search criteria are as for SearchCustomer above, but with wildcards not permitted – typically an email address is used to identify a customer, but this is not compulsory. Unlike SearchCustomer, a password is expected in the search criteria.

Example calling sequence

```
XElement search=new XElement("Customer",
    new XElement("Property", new XAttribute("Name","Name"), new
XAttribute("Value","Luke Harris")),
    new XElement("Property", new XAttribute("Name","Password"), new
XAttribute("Value","secret"))
    );
result = SynthesysCRM.AuthenticateCustomer(Token, "LUKE",
```

Example return is the string of the authenticated customer.
"LUKE_1"

UpdateCustomer, InsertCustomer, InsertUpdateCustomer

```
string InsertCustomer(string AuthenticationString, string Customers)
string UpdateCustomer(string AuthenticationString, string Customers)
string InsertUpdateCustomer(string AuthenticationString, string
Customers)
```

Input is a <Customers> document. Output is a <Customers> document with just customer Ids present in the same order, useful if you need to reference the new customer Id in subsequent operations.
The three flavours functions work similarly, except that :

- UpdateCustomer must have an existing customer ID present in the definition; this customer record will be updated with just those fields presented, other fields are left untouched.
- InsertCustomer must either not specify Customer Id (in which case one is generated) or must specify a new Customer Id.
- InsertUpdateCustomer is identical to UpdateCustomer if the customer Id is given and is in the database, otherwise it performs the same as InsertCustomer.

All three functions accept multiple customer records (which may even be for different customer prefixes); this can be useful for batch processing, although note that this function is transactional, any errors causing a rollback for just one customer will rollback the entire transaction.

Example calling sequence :

```
XElement xcustomernew = new XElement("Customers",
    new XElement("Customer", new XAttribute("Prefix", "LUKE"),
        new XElement("Property", new XAttribute("Name", "Customer
ID"), new XAttribute("Value", "LUKE_1")),
        new XElement("Property", new XAttribute("Name", "Name"), new
XAttribute("Value", "Luke Harris")),
        new XElement("Property", new XAttribute("Name",
"Enumeration"), new XAttribute("Value", "Blue"))
        ));
result = SynthesysCRM.UpdateCustomer(Token, xcustomernew.ToString());
```

Example return value :

```
<Customers>
  <Customer Prefix="LUKE">
    <Property Name="Customer ID" Value="LUKE_1" />
  </Customer>
```

```
</Customers>
```

GetCustomerHistory

```
string GetCustomerHistory(string AuthenticationString, string Prefix,
string CustomerId)
```

Gets history for a customer, including all notes. Includes document entries, but without the document contents (for efficiency), which should be retrieved separately.

Example call :

```
result = SynthesysCRM.GetCustomerHistory(Token, "LUKE", "LUKE_1");
```

Example return :

```
<History Prefix="LUKE" CustomerId="LUKE_1">
  <HistoryItem Type="Webflow">
    <object_index>3</object_index>
    <Performer>Luke</Performer>
    <DateTime>2009-07-31T11:53:21</DateTime>
    <Duration>45.937</Duration>
    <Direction>In</Direction>
    <Aborted>false</Aborted>
    <Result>Finish</Result>
    <Account>Luke</Account>
    <Campaign>TestGroups</Campaign>
  </HistoryItem>
  <HistoryItem Type="Customer Created">
    <object_index>2</object_index>
    <Performer>Luke</Performer>
    <DateTime>2009-07-31T11:54:02</DateTime>
  </HistoryItem>
  <HistoryItem Type="Webflow">
    <object_index>24</object_index>
    <Performer>Luke</Performer>
    <DateTime>2009-09-14T10:55:59</DateTime>
    <Duration>15382.88</Duration>
    <Direction>In</Direction>
    <Aborted>true</Aborted>
    <Result>OTHER</Result>
    <Account>Luke</Account>
    <Campaign>TestGroups</Campaign>
  </HistoryItem>
  <HistoryItem Type="Customer Details Updated">
    <object_index>22</object_index>
    <Performer>Luke</Performer>
    <DateTime>2009-09-14T10:56:28</DateTime>
  </HistoryItem>
  <HistoryItem Type="Customer Details Updated">
    <object_index>23</object_index>
    <Performer>Luke</Performer>
    <DateTime>2009-09-14T14:23:16</DateTime>
  </HistoryItem>
  <HistoryItem Type="Note">
    <object_index>26</object_index>
    <Performer>Luke</Performer>
    <DateTime>2009-09-15T22:34:42</DateTime>
    <Text>This is the note.</Text>
  </HistoryItem>
</History>
```

AddCustomerHistoryv4
This API is deprecated and may be removed in future versions, please contact Noetica if you wish to use this API.

AddCustomerNote
```
void AddCustomerNote(string AuthenticationString, string Prefix,
string CustomerId, string Note)
```
Adds a note to the customer history.

Example call :
```
SynthesysCRM.AddCustomerNote(Token, "LUKE", "LUKE_1", "A sample
note.");
```

AddCustomerDocument
```
void AddCustomerDocument(string AuthenticationString, string Prefix,
string CustomerId, string DocumentDetails)
```

DocumentDetails is an XML fragment with the following structure :
```
<Document>
<Base64Data>...Base64 encoded data of the document...</Base64Data>
<MimeType>application/pdf</MimeType>
<FileName>Invoice.pdf</FileName>
</Document>
```
Other elements may also be added and will be round-tripped (see GetCustomerDocument), but aren't required (for example document creation dates, author or other meta data).
Document data can be encoded using System.Convert.ToBase64String if using Microsoft.NET, other web development environments provide similar capabilities.

Sample calling sequence :
```
FileInfo fileInfo = new FileInfo(@"\\martini\e\Training On Line
Help\Draft OnLine Help Manual.Net\Synthesys.Net Help Manual\Training
SynthesysNet\1 SynthesysNet Overview.pdf");
byte[] byteData;
using (FileStream fileStream = fileInfo.OpenRead())
{
    byteData = new byte[(int)fileStream.Length];
    fileStream.Read(byteData, 0, (int)fileStream.Length);
}
string FileName = "1 SynthesysNet Overview.pdf";
string MimeType = "application/pdf";
string base64Data = System.Convert.ToBase64String(byteData);
XElement el = new XElement("Document",
    new XElement("FileName", FileName),
    new XElement("MimeType", MimeType),
    new XElement("Base64Data", base64Data));
SynthesysCRM.AddCustomerDocument(Token, "LUKE", "LUKE_1",
el.ToString());
```

GetCustomerDocument
```
string GetCustomerDocument(string AuthenticationString, string
Prefix, string CustomerId, int DocumentId)
```

Retrieves a customer document. The return value is a <Document> element as described above. The DocumentId is that found in the 'object_index' field as returned in the GetCustomerHistory function call.

Sample calling sequence :

```
result = SynthesysCRM.GetCustomerHistory(Token, "LUKE", "LUKE_1");

// Get any documents.
foreach (XElement historyItem in
XElement.Parse(result).Elements("HistoryItem"))
{
    if ((string)historyItem.Attribute("Type") == "Document")
    {
        int DocumentId =
int.Parse((string)historyItem.Element("object_index"));
        string document = SynthesysCRM.GetCustomerDocument(Token,
"LUKE", "LUKE_1", DocumentId);
        XElement XDocument = XElement.Parse(document);
        Console.WriteLine("Got document {0}",
XDocument.Element("FileName").Value);
    }
}
```

Synthesys CRM notes

Enumerations
Synthesys CRM stores these as integers in the database; the API looks after details of converting these to/from strings.

Booleans
Synthesys CRM stores these as integers (1 for true, 0 for false). The API converts these to True or False.

Active flag
To implement an Active flag in your CRM, add a Boolean property called 'Active' to the CRM. If this is present, then the API will automatically treat this as a marker of Active/Inactive records, and Inactive records will not be returned from searches unless specifically requested (by search criteria).
Additionally, calls to 'InsertCustomer' or 'InsertUpdateCustomer' that don't specify a value for 'Active' will assume the customer should be active and add this value automatically.

Passwords and salts
To implemented password hashes (SHA256) with randomised salts, then add a string called 'Password' with length of 44, and a 'Salt' string with length of 8. If these are present, then the API will automatically allow password protection on CRM records, in particular :

1. The AuthenticateCustomer API will be available.
2. On inserting/updating a record, any supplied password will be hashed with a randomised salt.
3. On retrieving a record, the password will be replaced by five stars. If this value is seen in the insert/update of a record, it will be ignored.
4. Note that the original password is not stored.

XML Notes

Customer Description
Example customer description

```
<CustomerDescription Name="LUKE">
  <Property Id="P001" Name="Customer ID" Type="String"
IsCustomerId="True" />
  <Property Id="P002" Name="Name" Type="String" IsName="True"
Maximum="40" IsIndexed="True" />
  <Property Id="P003" Name="Telephone" Type="String"
IsTelephone="True" Maximum="40" IsIndexed="True" />
  <Property Id="P004" Name="Enumeration" Type="String"
IsEnumeration="True" IsRequired="True">
    <Choice Name="Red" />
    <Choice Name="Yellow" />
    <Choice Name="Orange" />
    <Choice Name="Blue" />
  </Property>
  <Group Id="P005" Name="Group">
    <Property Id="P006" Name="Field1" Type="String" Maximum="40" />
    <Property Id="P007" Name="Field2" Type="String" Maximum="40" />
    <Property Id="P008" Name="Field3" Type="String" Maximum="40" />
  </Group>
  <Group Id="P009" Name="Noetica Address Control" IsAddress="True">
    <Property Id="P010" Name="Line1" Type="String" Maximum="56"
IsAddress="True" />
    <Property Id="P011" Name="Line2" Type="String" Maximum="56"
IsAddress="True" />
    <Property Id="P012" Name="Line3" Type="String" Maximum="56"
IsAddress="True" />
    <Property Id="P013" Name="Line4" Type="String" Maximum="56"
IsAddress="True" />
    <Property Id="P014" Name="Line5" Type="String" Maximum="56"
IsAddress="True" />
    <Property Id="P015" Name="Line6" Type="String" Maximum="56"
IsAddress="True" />
    <Property Id="P016" Name="PostCode" Type="String" Maximum="8"
IsAddress="True" IsPostcode="True" />
  </Group>
  <Property Id="P017" Name="Pet Name" Type="String" Maximum="40"
IsMultiline="True" IsRequired="True" />
  <Property Id="P018" Name="DateTime" Type="DateTime" />
  <Property Id="P019" Name="Number" Type="Numeric" />
  <Property Id="P020" Name="BoolType" Type="Boolean" />
  <Property Id="P021" Name="Active" Type="Boolean" IsRequired="True"
IsActive="True" />
  <Property Id="P022" Name="Password" Type="String" Maximum="44"
IsPassword="True" />
  <Property Id="P023" Name="Salt" Type="String" Maximum="8"
IsSalt="True" />
</CustomerDescription>
```

Explanation of attributes :

Name	Default	Explanation
Id		Internal Id of the field. For Synthesys CRM this is

		currently the 'P' number e.g. P001 for customer Id. This can be ignored in most scenarios, use the Name field instead.
Name	(Required)	Name of the field. Typically should be translated before display. For Synthesys CRM this is unique.
Type	(Required)	String, Numeric, Boolean, DateTime
IsCustomerId	False	Is this field the customer ID? Although this is commonly called 'Customer ID', it is possible that this field can be renamed.
IsPassword	False	Is this a (hashed) password field?
IsSalt	False	Is this a salt for the password
IsActive	False	Is this a Boolean field representing 'active' status of the customer?
IsPrimary	False	Is this the primary contact of the customer? Not currently implemented
IsTelephone	False	Field represents a telephone number.
IsEmail	False	Field represents an email address.
IsName	False	Field represents the name or part of the name
IsAddress	False	Field represents the address or part of the address (including postcode).
IsPostcode	False	Field represents the postcode or part of the postcode. Not currently implemented.
Minimum	-	Minimum length of the field. Not currently implemented
Maximum	-	Maximum length of the field.
IsMultiline	False	If True, string value will contain \r\n separators to indicate multiple lines of text.
IsRequired	False	Is the field mandatory?
IsEnumeration	False	Is this an enumerated type? Any type listed as an 'enumeration' will be followed with multiple 'Value' elements, each of which is one of the acceptable values for the enumeration.
IsUnique	False	Field is unique. Not currently implemented.
IsIndexed	False	Field is indexed.

Customer Data

For simplicity and flexibility, we base most input/output customer requests on an XML structure containing as root a <Customers> element, and zero or more sub <Customer> elements within that. This means that many of the API functions will in fact be able to deal with multiple requests at once; so 'create customer' can process 100 customer creations in one go if required. There is no need to use the API in this way, but it can be more efficient for bulk loading of data.

Example

```
<Customers>
<Customer Prefix="ELECT">
<Property Name="CustomerId" Value="ELECT_01" />
<Property Name="Surname" Value="Harries" />
...
</Customer>
```

```
...
</Customers>
```

If Value is missing, then it implies a database NULL. Booleans are presented as strings, ‘True’ or ‘False’. Enumerations have one of the enumerated values as a string.