finPOWER Connect 3
Web Services Connectivity and Programming Guide

Version 3.03

25th May 2020

Table of Contents

Disclaimer

This document contains information that may be subject to change at any stage.

All code examples are provided "as is".

Copyright Intersoft Systems Ltd, 2020.

Version History

Date	Version	Name	Changes
07/03/2014	1.00	PH	Created.
17/03/2014	1.01	PH	Clarified that Web Services cannot currently be used for accessing external resources.
20/03/2014	1.02	PH	Added serialisation/ deserialisation section and samples.
09/04/2014	1.03	PH	Added dates section.
02/05/2014	1.04	PH	Updated information regarding session tokens and when to re-authenticate.
06/05/2014	1.05	PH	Clarified section detailing session token in Authorization header.
06/06/2014	1.06	PH	Added troubleshooting guide for address database issues and PDF sections.
21/07/2014	1.07	PH	Updated ExecuteRequest sample.
21/07/2014	1.08	PH	Support for external Web Services detailed in Limitations section.
12/08/2014	1.09	PH	Custom Web Services section enhanced.
13/08/2014	1.10	PH	Custom Web Service Enum deserialisation.
01/10/2014	1.11	PH	Custom Web Service section used to note new "Execute (Parameters)" option from Test Web Service form.
30/03/2015	1.12	PH	Added link to Microsoft article in Introduction, Web Services section.
14/04/2015	1.13	PH	Split out Custom Web Service related sections into a new finPOWER Connect Custom Web Services Programming Guide document.
21/04/2015	1.14	PH	Added Loan Application 1 (Visual Basic) sample documentation.
30/04/2015	1.15	PH	Added sample JSON serialisation and deserialisation code.
22/07/2015	1.16	PH	Updated LoanApplication1VB sample.
18/08/2015	1.17	PH	Added AuthenticateUserToken Web Service.
30/09/2015	1.18	PH	Mention new Web Subscriber IP White List.
12/07/2016	3.00	PH	Updated for finPOWER Connect version 3.
19/02/2019	3.01	PH	Updated screen shots and incorrect PDF title.
10/02/2020	3.02	PH	Updated.
25/05/2020	**3.03**	**PH**	**Updated for new POST authentication methods.**

Introduction

This document is intended for software developers who would like to connect to the finPOWER Connect Web Services over HTTP.

This document provides all the necessary information needed to connect to and perform requests against the Web Services.

For information on installing and configuring the finPOWER Connect Web Services on a Web Server, see the **finPOWER Connect 3 Web Services Installation and Configuration** document.

NOTE: finPOWER Connect HTML Widgets and Portals both provide a model which makes it easy to produce User Interfaces and use the finPOWER Connect business layer.

Consider these unless you must use Custom Web Services (e.g., to access finPOWER Connect from a native App or an existing Website).

Limitations

Where access to a resource external to the finPOWER Connect database is required and a built-in Web Service does not exist to achieve this, it should be assumed that such access cannot currently be achieved and therefore any **Custom Web Services or Scripts that run within finPOWER Connect as part of a Web Service call** cannot access these resources.

This includes the following:

- Document Manager
  - Accessing the Document Manager folder structure is not possible in the usual manner from a Web Service.
- Document publishing involving Word or Excel VBA to create a document.
  - Microsoft Office should never be run on a Web Server and therefore only a limited form of Document publishing is available from a Web Service.
  - **NOTE:** As of finPOWER Connect 2.02.06, the concept of 'Unattended Publishing' was introduced. This is covered in the finPOWER Connect 3 Custom Web Services Programming Guide document.
- Access to external Web Services, e.g.:
  - Credit Bureau (Centrix, DecisionLogic etc)
  - MotorWeb

WARNING: Any Custom Web Services or Scripts that attempt to access external resources cannot be supported and cannot be guaranteed to work in future releases of finPOWER Connect and the finPOWER Connect Web Services.

Unless listed in the table below, assume the external resource is not supported.

The following external Web Services are supported. The Web Services version at which support was added is detailed below:

Service	Version	Details
Credit Sense	2.01.01.00	Comprehensive Web Service support.
Veda (Australia)	2.01.01.00	Limited Web Service support.
PPSR G2B (New Zealand)	2.02.02.00	Limited Web Service support.
Edentiti GreenId	2.02.05.00	Moderate Web Service support.

NOTE: Many external Web Services have only limited finPOWER Connect Web Services support since it is anticipated that Custom Web Services will more likely be used to access the external service.

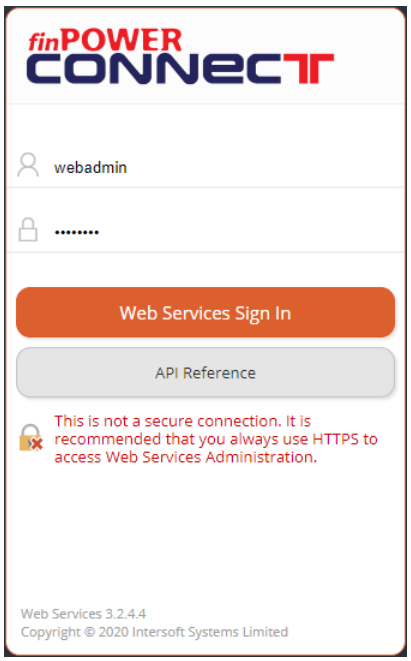
System Requirements and Prerequisites

- Since the finPOWER Connect Web Services use HTTP, the only programming requirement is a development environment that allows HTTP and secure HTTP (HTTPS) requests to be made.
  - The application being developed can use any programming language and exist on any platform providing the above is supported.
- A connection to a Web Server running the finPOWER Connect Web Services is required.
  - This Web Server might be:
    - Accessible over the Internet.
    - Accessible over a private LAN.
    - Running on the development PC (for testing purposes).
      - The **finPOWER Connect 3 Web Services Installation and Configuration** document details installation of the Web Services for testing purposes.
- From a development point-of-view, a solid knowledge of the following is recommended:
  - HTTP connectivity.
  - XML and/ or JSON.
- Most code samples are currently limited to Visual Basic (VB.NET).

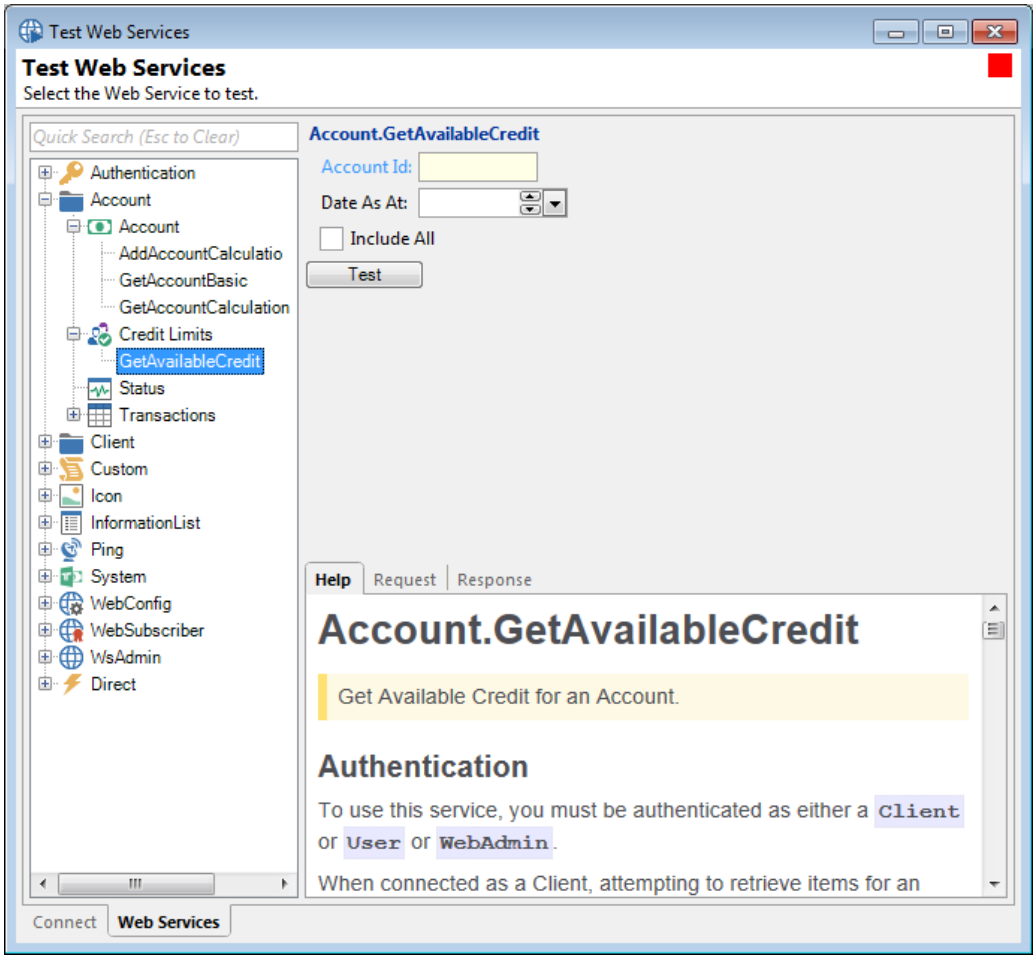
Help Resources

The finPOWER Connect Web Services have online help detailing all Web Services. This help is available from:

- The finPOWER Connect Web Services Web Server via the **API reference** button on the Sign In page:



- From within the finPOWER Connect Windows User Interface via the **Test Web Services** form (Tools, Web, Test Web Services).



  - The help is available by selecting a Web Service on the **Web Services** tab, providing a valid **URL** has been entered on the **Connect** page.
- This help is constantly evolving and is updated for every new Web Service that is added.

Web Services

- All Web Services are based on the Microsoft Web API framework meaning:
  - They use HTTP/ HTTPS as a transport protocol.
  - They do not use SOAP and therefore no WSDL is available.
    - XML (or JSON) can be parsed manually or, if using a language that supports it (e.g., VB.NET), deserialised into a simple object.
  - They use a RESTful, RPC based approach meaning:
    - Most services are fully URL-based, e.g., to retrieve a list of Accounts for a finPOWER Connect Client, you would simply request a URL such as:
      - `/Api/Client/GetAccounts?clientId=C10000&includeQuote=true`
  - Unless otherwise specified, both XML and JSON are supported.
    - Generally, only the HTTP response will contain XML/ JSON data but services that require more than just simple URL parameters can POST either XML or JSON.
- The finPOWER Connect Web Services consist of many individual services.
  - These services are organised into 'Controllers' which is simply a way of grouping related services, e.g., the above URL example calls the **GetAccounts** service which is located in the **Client** controller.
    - **NOTE:** The Web Services help and the Test Web Services form within finPOWER Connect organise their table of contents with a higher level of grouping for readability, e.g., both **Client** and **ClientWebMail** controllers are grouped within a **Client** folder.

The following are useful articles for understanding Microsoft's Web API and REST services:

http://blogs.msdn.com/b/martinkearn/archive/2015/01/05/introduction-to-rest-and-net-web-api.aspx

Connectivity and Security

- Connectivity to the Web Services is via HTTP/ HTTPS.
  - HTTP is fine for testing but secure HTTP (HTTPS) MUST be used in a production environment.
  - The HTTP link to Web Services may be over a private network or the Internet (public) or, to an installation of the Web Services running on the same PC (usually for development purposes only, using localhost).

Encryption

- All communication (in a production environment) takes place over a secure, encrypted HTTP channel using SSL/ TLS.
  - This is enabled by a certificate installed on the Web Server hosting the Web Services and is outside of the scope of this document.

Authentication

- Most Web Services require authentication.
- Any application wishing to access the Web Services must have a **Web Subscriber** record defined in the finPOWER Connect database.
  - This record has an **Id** and a **Secret key**, both of which are required for authentication.
- Authenticated services must be passed a special **Session Token** as an HTTP header. This is returned from the initial authentication request.
  - The Session Token is valid for up to 24 hours.

WARNING: Authentication methods support both GET and POST.

If you are using the GET, IIS is more likely to record sensitive information in its log files (depending on the server's configuration) since credentials are included in the URL.

Therefore, POST is the recommended method since it is less likely to be logged and is less easily readable.

Setting up a Test Environment

When programming against the finPOWER Connect Web Services, it is useful to be able to test the Web Services without having to write any code.

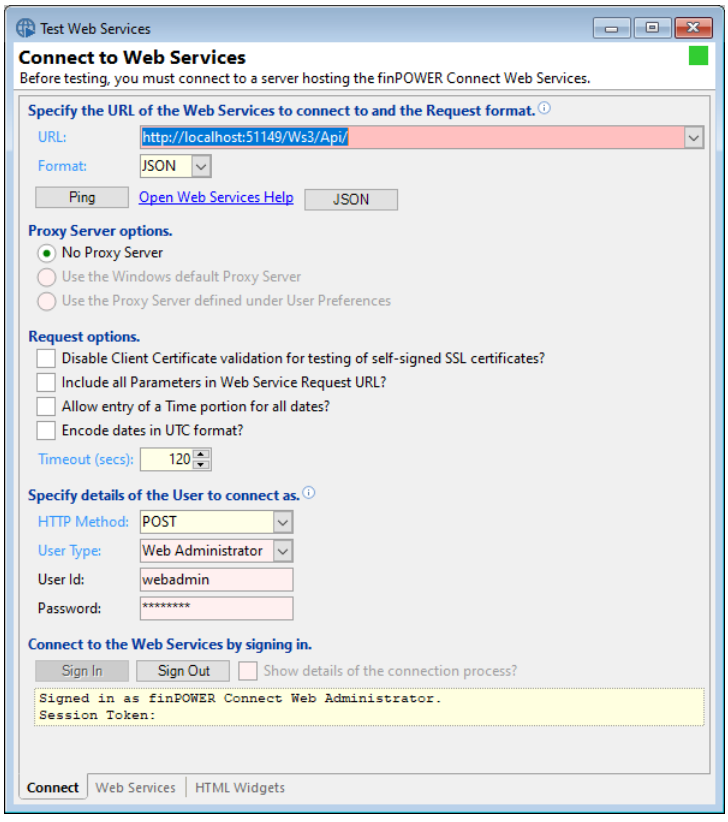
finPOWER Connect provides a **Test Web Services** form for this purpose.

The following steps are required to set up a test environment.

NOTE: This assumes you can connect to a Web Server running the finPOWER Connect Web Services. Installation and configuration of the Web Services is detailed in the **finPOWER Connect 3 Web Services Installation and Configuration** document.

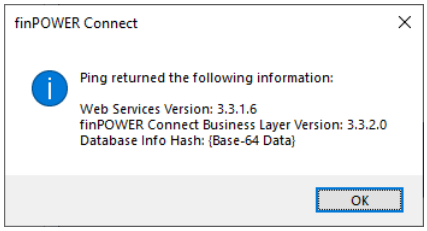
Setup and Connection

- Install the latest version of finPOWER Connect.
- Open finPOWER Connect.
  - If available, open the finPOWER Connect database to which the Web Services connect.
    - This is not necessary but does make the connection and testing process easier, e.g., if you call a Web Service to add an Account, you can then view the Account directly from within finPOWER Connect.
  - If unavailable, open any finPOWER Connect database, e.g., the demonstration database.
- Ensure you are logged in as an administrator User.
- From the **Tools** menu, select **Web**, **Test Web Services**.
- On the **Connect** page, enter the URL of the Web Services to access, e.g.



NOTE: The Web Services URL will always end with **/Api/**.

- Click the **Ping** button to test that the Web Services are available.
- All going well, you should see a message like this:

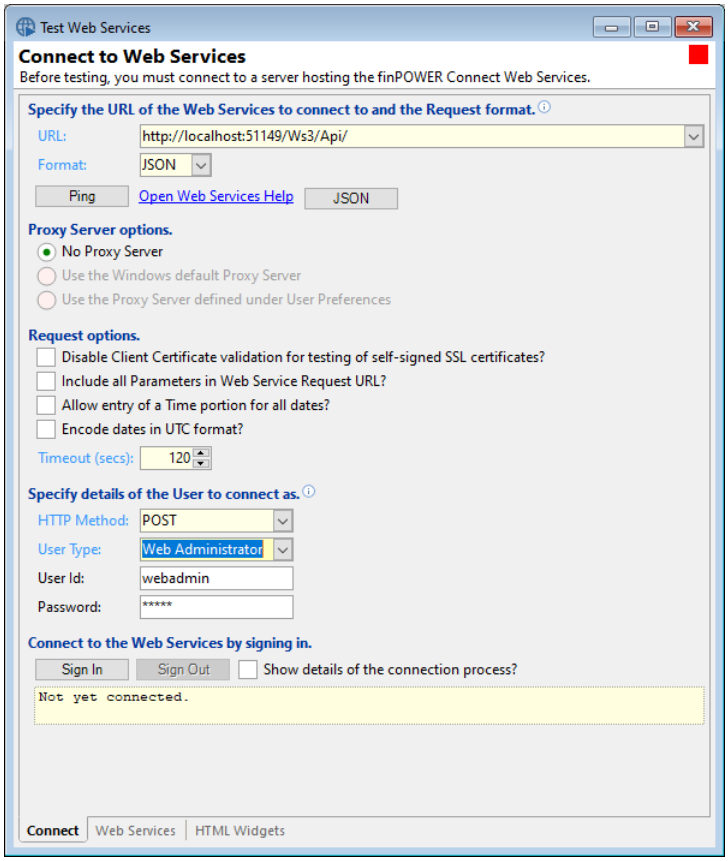


- NOTE: Ping is one of the few Web Services that do not require authentication.

- Now you can connect to the Web Services by specifying details of the user to connect as. The **User Type** determines how you connect.
  - Note that connecting as a finPOWER Connect User or a Client requires the Id and Secret Key of a Web Subscriber to connect as.
  - If you have the finPOWER Connect Web Services database loaded, you can add a new Web Subscriber record (if required) as follows:
    - From the **Tools** menu, select **Web**, **Web Subscribers**.
    - Click the **Add** button and enter the following:
      - Code: TEST
      - Name: Test Web Services
    - Click the **Save** button.
    - Close the **Web Subscribers** form.
  - If you do not have the finPOWER Connect Web Services database, you will need to contact the database administrator and ask them to set up a Web Subscriber record for you to use and to supply you with the Web Subscriber Id and Secret Key.

Web Administrator

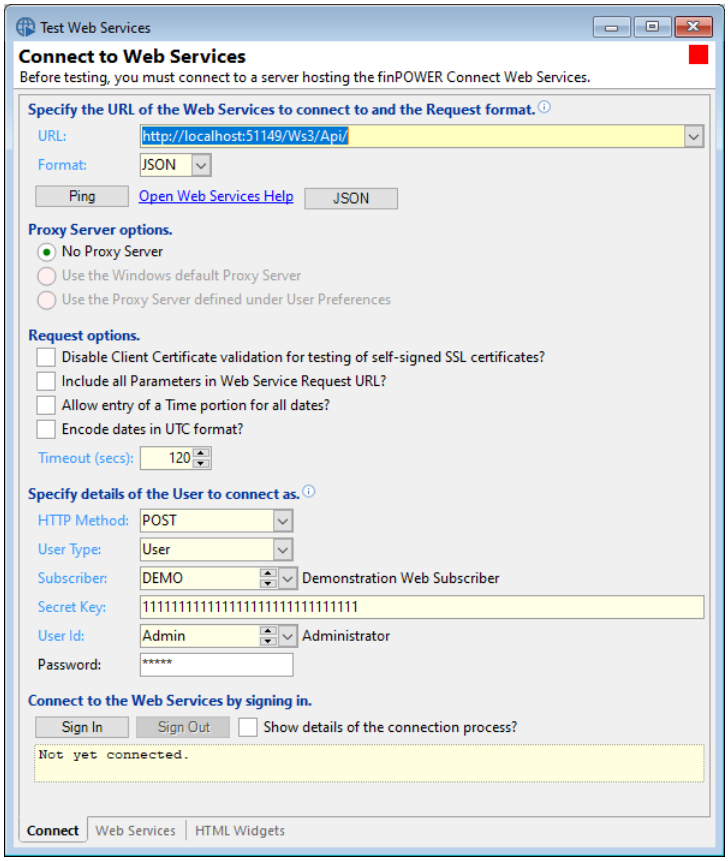
- Connecting as a Web Administrator requires only the Web Administration User Id and Password. By default these are:
  - User Id: webadmin
  - Password: password



NOTE: The Web Administrator role is for use in the Web Services Administration facility and would not generally be used by external applications.

User

- Connecting as a finPOWER Connect User requires both the User Id and Password and also the Id and Secret Key of a Web Subscriber to connect as.

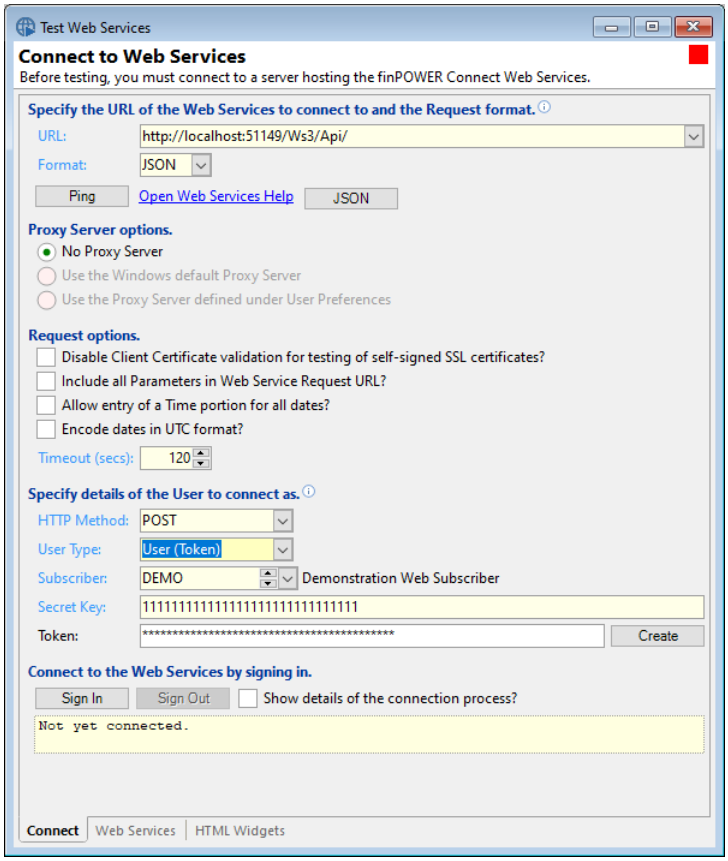


- If you have the Web Services database open, you can select the Subscriber from the dropdown. If not, you will need to be enter the details manually in the **Subscriber** dropdown and **Secret Key** fields.
- Enter the credentials of a finPOWER Connect User, e.g.
  - User Id: admin
  - Password: admin

NOTE: The User must have been granted Web Access. This is configured within finPOWER Connect via the **Web Access** page on the **Users** form (**Tools**, **User Security**, **Users**).

User (Token)

- Connecting as a finPOWER Connect User using a token requires a finPOWER Connect-generated Token and also the Id and Secret Key of a Web Subscriber to connect as.



- If you have the Web Services database open, you can select the Subscriber from the dropdown. If not, you will need to be enter the details manually in the **Subscriber** dropdown and **Secret Key** fields.
- Click the "Create" button to create a Token.

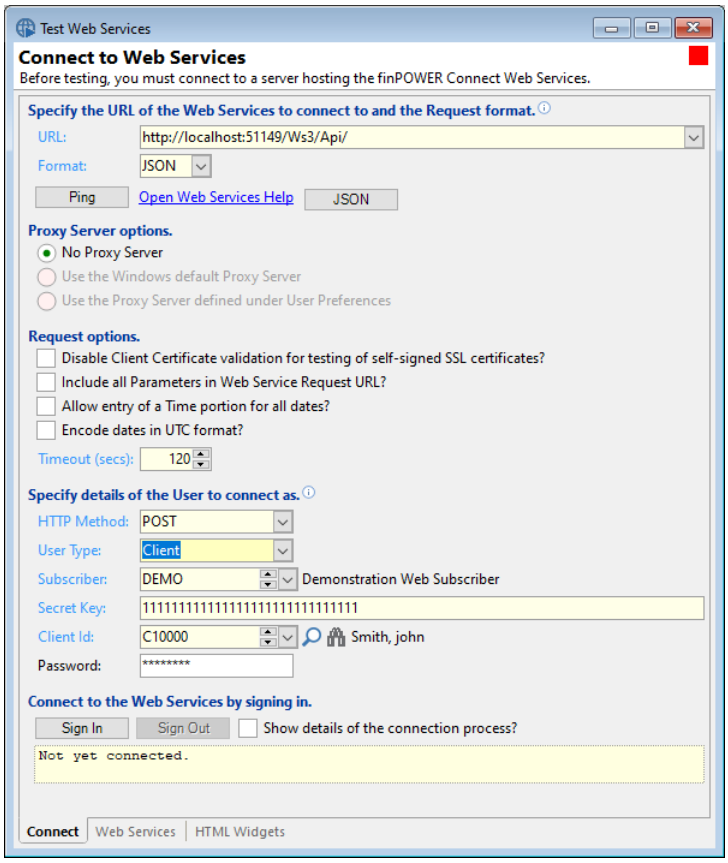
NOTE: The User must have been granted Web Access. This is configured within finPOWER Connect via the **Web Access** page on the **Users** form (**Tools**, **User Security**, **Users**).

Also, Token-based logins must be enabled for the database (they are disabled by default) using the Tools, Global Settings, Web, General page.

WARNING: Unlike the other authentication methods, connecting to Web Services using a token requires that the database showing the Test Web Services form is the same one being used by the Web Services.

Client

- Connecting as a finPOWER Connect Client requires both the Client Id and Password and also the Id and Secret Key of a Web Subscriber to connect as.

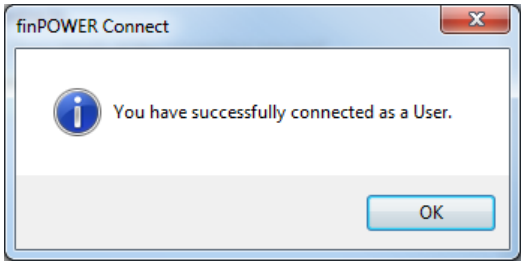


- If you have the Web Services database open, you can select the Subscriber from the dropdown. If not, you will need to be enter the details manually in the **Subscriber** dropdown and **Secret Key** fields.
- Enter the credentials of a finPOWER Connect Client, e.g.
  - Client Id: C10000
  - Password: Password1

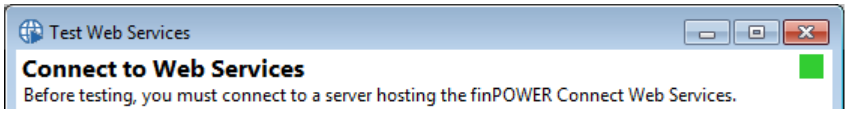
NOTE: The Client must have been granted Web Access. This is configured within finPOWER Connect via the **Web Access** page on the **Clients** form (**Client**, **Clients**).

Signing In

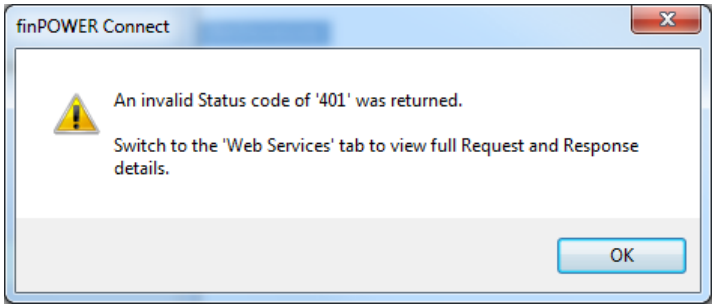
- After entering the Web Administrator/ User/ Client details, click the **Connect** button to sign in.
- You should see a message similar to this:



And a green box will appear in the top-right corner of the form to show that you have connected successfully:



- If signing in was unsuccessful, you will see a message such as:



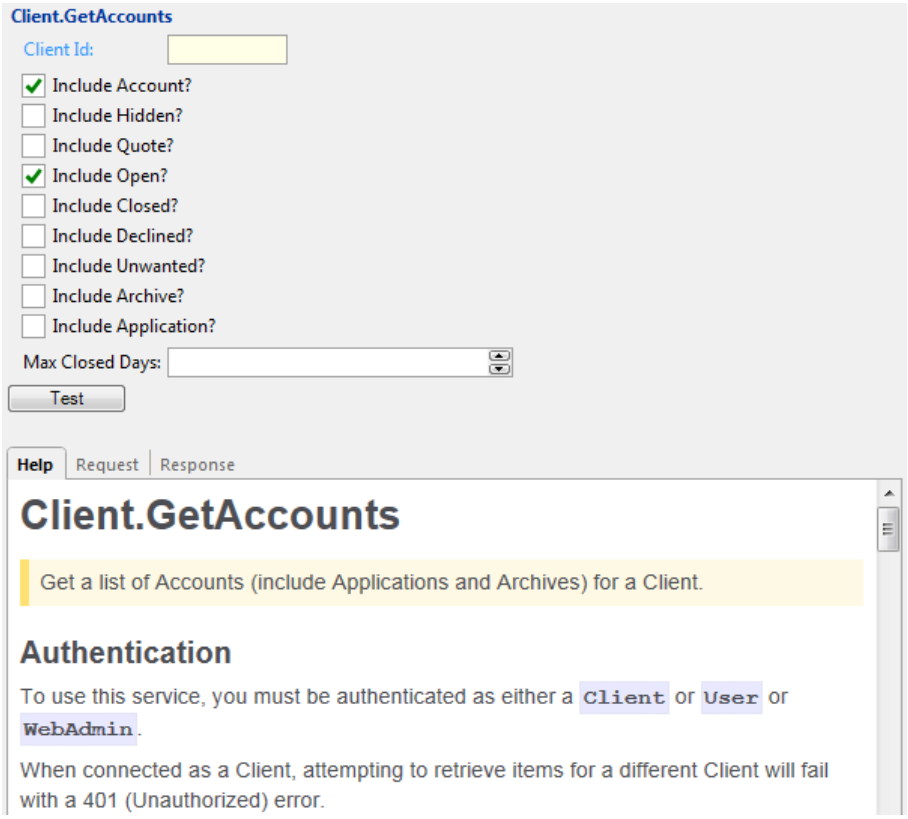
  - Click **OK** and switch to the **Web Services** tab.
  - Click the **Response** tab.
  - The response should contain an XML error message which should help you determine why signing in failed. The following example has removed attributes on the Error tag for clarity):

```
<Error>
  <Message>Failed to authenticate User.</Message>
  <Code>User.InvalidCredentials</Code>
  <InternalMessage>Failed to Authenticate User.</InternalMessage>
</Error>
```

Using the Web Services Test Form

Now that you have successfully connected to the Web Services, you can test any Web Service or view online help as follows.

- Switch to the **Web Services** tab.
- Locate and select the Web Service to test in the explorer.
  - You can use the **Quick Search** box above the explorer to filter Web Services.
- The **Help** tab allows you to view the selected Web Service's help.
- The **Request** and **Response** pages allows you to view the information sent to and received from the Web Services.
- For example:
  - Type **Client Accounts** into the Quick Search box.
  - Select the **Client**, **GetAccounts** Web Service.
  - A list of parameters that this Web Service accepts is displayed along with help for the Web Service, e.g.



  - Enter any parameters, e.g.
    - Client Id: C10000
    - Include Quote: checked
  - Click the **Test** button.
  - The **Response** tab will automatically be selected and the results of the test will be displayed, e.g.

- By default, the response will be retrieved as XML. This can be changed from the **Connect** page by changing the **Format** to **JSON**.

Programming Languages

You may develop your application in the programming language of your choice, on the platform of your choice. The majority of the code samples throughout this document however use Visual Basic (VB.NET).

Most finPOWER Connect Web Services allow you to use either XML or JSON for transferring data.

NOTE: No formal XML or JSON schemas are defined and the XML/ JSON responses may be extended over time hence any parsing code should assume that the response may contain additional nodes/ properties in the future.

Microsoft .NET

Microsoft .NET programming languages (Visual Basic and C#) allow the response returned from a Web Service to be parsed in different ways, e.g., you may wish to parse an XML response using an `XmlDocument` object or you could use .NET's built-in deserialisation to automatically convert the XML (or JSON) into a simple .NET object.

The following are Visual Basic code examples of parsing an XML error response of:

```
<Error xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Message>Failed to authenticate User.</Message>
  <Code>User.InvalidCredentials</Code>
  <InternalMessage>Failed to Authenticate User.</InternalMessage>
</Error>
```

Parsing using the XmlDocument

```
Private Sub ParseXml(xml As String)

  Dim Document As System.Xml.XmlDocument
  Dim Node1 As System.Xml.XmlNode
  Dim Node2 As System.Xml.XmlNode

  Dim Message As String
  Dim Code As String
  Dim InternalMessage As String

  ' Load XML Document
  Document = New System.Xml.XmlDocument()
  Document.LoadXml(xml)

  ' Get Root <Error> node
  Node1 = Document.SelectSingleNode("//Error")

  ' Get properties
  Node2 = Node1.SelectSingleNode("Message")
  If Node2 IsNot Nothing Then Message = Node2.InnerText
  Node2 = Node1.SelectSingleNode("Code")
  If Node2 IsNot Nothing Then Code = Node2.InnerText
  Node2 = Node1.SelectSingleNode("InternalMessage")
  If Node2 IsNot Nothing Then InternalMessage = Node2.InnerText

End Sub
```

Deserialisation into a .NET Object

```
Private Sub ParseXml(xml As String)

  Dim ErrorDetails As WSErrorDetails
  Dim ms As System.IO.MemoryStream
  Dim Obj As Object
```

```
  Dim XmlSerializer As Serialization.XmlSerializer

  ' Deserialise
  Try
    ' Create Serialiser
    XmlSerializer = New Serialization.XmlSerializer(GetType(WSErrorDetails))

    ' Deserialise
    ms = New System.IO.MemoryStream(Encoding.UTF8.GetBytes(xml))
    ErrorDetails = DirectCast(XmlSerializer.Deserialize(ms), WSErrorDetails)
  Catch ex As Exception
    ' Failed
  Finally
    If ms IsNot Nothing Then ms.Close(): ms.Dispose()
  End Try

End Sub

<Xml.Serialization.XmlType("Error")>
Public Class WSErrorDetails
  Public Message As String
  Public Code As String
  Public InternalMessage As String
End Class
```

A more comprehensive example of deserialisation is given in the Deserialisation section.

Authenticating (Signing In)

This section details the authentication process.

Most Web Service require that a special 'Session Token' is included in an HTTP header. This Session Token is generated during the authentication process, itself a Web Service, and is valid for 24 hours.

Web Roles

A different authentication service is used depending on the type of user that you are signing in as. The type of user defines a 'Web Role' which determines which services can be used.

The Web Roles are:

- **User** (a finPOWER Connect User)
- **Client** (a finPOWER Connect Client)
- **WebAdmin** (the Web Services Administrator)

NOTE: You will only ever use User or Client since WebAdmin is used internally for administering Web Services.

Web Subscribers

For an application to use Web Services, that application must first have a Web Subscriber record defined in finPOWER Connect.

Web Subscribers are maintained in finPOWER Connect from the **Tools** menu, **Web**, **Web Subscribers**.

Each Subscriber is given a unique **Subscriber Id** and also a **Secret Key** which are used during the authentication process.

If successful, the authentication service returns a Session Token. This is tied to the Web Subscriber's Secret Key therefore, if the Web Subscriber's Secret Key is changed (via the Web Subscribers form), the Session Token will immediately become invalid.

NOTE: Web Subscribers can define an IP White List to ensure that requests are coming from a known IP Address, i.e., the IP Address of the Web Server hosting the Web Application.

The Authentication Process

When authenticating, the Subscriber's Secret Key is never sent across the network. It is used to produce a Hash which is sent with the authentication request.

An example GET request to authenticate a finPOWER Connect User is shown below:

```
GET https://
Ws3/Api/Authentication/AuthenticateUser?subscriberId=DEMO&userId=Admin&password=admin&hash=NxyOY%2BjZ
14QsT3OYJvdw0On9u3fj%2FH9VVVPiRq50EsM%3D&hashSalt=EMQMPSRCDP HTTP/1.1
Content-Type: application/json
Host: localhost:51149
```

The subscriberId, userId and password parameters are self-explanatory. The hash and hashSalt are explained in the next section.

The corresponding POST request packages these parameters up as JSON and therefore does not include them in the URL, e.g.:

```
POST http://localhost:51149/Ws3/Api/Authentication/AuthenticateUser HTTP/1.1
Content-Type: application/json; charset=UTF-8
Host: localhost:51149
Content-Length: 164
Expect: 100-continue

{
  "subscriberId": "DEMO",
  "userId": "Admin",
  "password": "admin",
  "hash": "C/xymFJ0VOsYPEHfrBL4f3BKSer6tkCxjn68T/nbJs8=",
  "hashSalt": "MGYVJPQYHI"
}
```

hashSalt

The hashSalt parameter is simply a string of 10 random letters. The following Visual Basic code sample forms a random string.

```
Rnd = New Random(CInt(Now.Ticks Mod Int32.MaxValue))
hashSalt = ""
For i = 1 To 10
  hashSalt &= Chr(65 + Rnd.Next(0, 25))
Next
```

hash

The hash parameter is an SHA256 hash produced by concatenating the Hash Salt, User Id, Password and the Web Subscriber's Secret Key. The following pseudo code shows the process of creating the hash:

```
s = hashSalt & userId & password & secretKey
hash = Sha256(s)
hash = Base64Encode(hash)
```

Authentication Code Sample

The following is a Visual Basic code example of signing in as a finPOWER Connect User. It demonstrates how to create the Hash required by the authentication service and how to send a request to the Web Services.

The resultant Session Token can then be used for performing authenticated requests.

The **ExecuteRequest** and **CreateHash** functions are taken directly from the Web Forms examples in the /Samples folder of the Web Services application.

```
' Connection Constants
Const WEB_SERVICES_URL As String = "http://localhost/finPOWERConnectWS2/Api/"
Const WEB_SUBSCRIBER_ID As String = "CC"
Const WEB_SUBSCRIBER_SECRET_KEY As String = "1F673CE613414"
Const USER_ID As String = "admin"
Const USER_PASSWORD As String = "admin"

Public Sub Test()

  Dim Document As System.Xml.XmlDocument
  Dim ErrorMessage As String
  Dim Hash As String
  Dim HashSalt As String
  Dim Node1 As System.Xml.XmlNode
  Dim Nodes As System.Xml.XmlNodeList
  Dim Ok As Boolean
  Dim RequestUrl As String
  Dim ResponseText As String
  Dim SessionToken As String
  Dim StatusCode As Integer

  ' Assume Success
  Ok = True
```

```
  ' Initialise
  Document = New System.Xml.XmlDocument()

  ' ------------------------------------------------
  ' Connect to Web Services as a finPOWER Connect User
  ' ------------------------------------------------
  If Ok Then
    ' Produce a hash (and a random hash salt) of the User Id and Password
    Hash = HashSha256(USER_ID & USER_PASSWORD, WEB_SUBSCRIBER_SECRET_KEY, HashSalt)

    ' Build Request URL
    RequestUrl = WEB_SERVICES_URL
    RequestUrl &= String.Format("Authentication/AuthenticateUser?subscriberId={0}&userId={1}&passwor
d={2}&hash={3}&hashSalt={4}", Server.UrlEncode(WEB_SUBSCRIBER_ID), Server.UrlEncode(USER_ID), Server
.UrlEncode(USER_PASSWORD), Server.UrlEncode(Hash), Server.UrlEncode(HashSalt))

    ' Execute Request
    Ok = ExecuteRequest(RequestUrl, "GET", "XML", "",
Nothing, "", ResponseText, StatusCode, ErrorMessage, Nothing) AndAlso StatusCode = HttpStatusCode.OK

    ' Parse Response
    If Ok Then
      ' Parse Response to retrieve Session Token
      Document.LoadXml(ResponseText)
      SessionToken = Document.SelectSingleNode("//SessionDetails/SessionToken").InnerText
    End If
  End If

  ' Error
  If Not Ok Then
    ' If ResponseText contains <Error>, this can be parsed as an XML document
  End If

End Sub

''' <summary>
'''  Execute a Request (sending either nothing, text or binary data in the form of a Byte array) to
retrieve either a text or binary (Byte array) Response.
''' </summary>
''' <param name="requestUrl">
'''  The Request URL.
''' </param>
''' <param name="httpMethod">
'''  The HTTP Method, e.g., GET or POST.
''' </param>
''' <param name="contentType">
'''  The Content Type for the HTTP Content-Type header or just 'XML' or 'JSON'.
''' </param>
''' <param name="requestText">
'''  The Request Text or a blank String if not sending any text in the body of the request or if
sending binary data using the requestBytes parameter.
''' </param>
''' <param name="requestBytes">
'''  The Request Bytes or Nothing if not sending binary data to the request.
''' </param>
''' <param name="sessionToken">
'''  The Session Token or a blank String is using a unauthenticated Web Service.
''' </param>
''' <param name="responseText">
'''  The Response Text. This will be Nothing if the content type of the Response did not indicate
that this was a text response, i.e., it did not begin 'text'.
''' </param>
''' <param name="responseBytes">
'''  The binary Response. This will be Nothing if the content type of the Response indicates a text
response.
''' </param>
''' <param name="statusCode">
'''  The HTTP Status Code received.
''' </param>
''' <param name="errorMessage">
'''  An Error Message. This will only by populated if this function returns False.
''' </param>
''' <param name="response">
'''  The Response object. May be useful for debugging purposes.
''' </param>
''' <returns>
'''  A Boolean value indicating success.
''' </returns>
''' <remarks>
'''  NOTE: This is a sample only and matches the function used in the Web Services Test form within
```

```
finPOWER Connect. This sample may be subject to updating at any time.
''' </remarks>
Private Function ExecuteRequest(requestUrl As String,
                                httpMethod As String,
                                contentType As String,
                                requestText As String,
                                requestBytes() As Byte,
                                sessionToken As String,
                                ByRef responseText As String,
                                ByRef responseBytes() As Byte,
                                ByRef statusCode As Integer,
                                ByRef errorMessage As String,
                                ByRef response As System.Net.HttpWebResponse) As Boolean

  Dim Buffer(1023) As Byte
  Dim Bytes As Integer
  Dim Encoding As System.Text.UTF8Encoding
  Dim Ok As Boolean
  Dim MemoryStream As System.IO.MemoryStream
  Dim Request As System.Net.HttpWebRequest
  Dim Stream As System.IO.Stream
  Dim srText As System.IO.StreamReader

  ' Assume Success
  Ok = True

  ' Initialise ByRef Parameters
  responseText = ""
  responseBytes = Nothing
  statusCode = 0
  errorMessage = ""
  response = Nothing

  ' Initialise
  Select Case UCase(contentType)
    Case "JSON" : contentType = "application/json"
    Case "XML" : contentType = "text/xml"
  End Select

  ' Create Web Request
  Try
    Request = DirectCast(System.Net.WebRequest.Create(requestUrl), System.Net.HttpWebRequest)
    With Request
      ' General
      .Method = httpMethod
      .ContentType = contentType
      .Timeout = 20000 ' 20 Seconds

      ' Add authorisation header
      If Len(sessionToken) <> 0 Then
        .Headers.Add("Authorization", String.Format("AuthFinWs token=""{0}""", sessionToken))
      End If

      ' Write Request Body
      If Len(requestText) = 0 AndAlso requestBytes Is Nothing Then
        ' None
        .ContentLength = 0
      Else
        ' Text/ Binary
        Try
          ' Encode Text as UTF8
          If Len(requestText) <> 0 Then
            .ContentType &= "; charset=UTF-8"
            Encoding = New System.Text.UTF8Encoding()
            requestBytes = Encoding.GetBytes(requestText)
          End If

          ' Content Length
          .ContentLength = requestBytes.Length

          ' Content
          Stream = .GetRequestStream()
          Stream.Write(requestBytes, 0, requestBytes.Length)
        Catch ex As Exception
          Ok = False
          errorMessage = ex.Message
        Finally
          If Stream IsNot Nothing Then
            Stream.Close()
            Stream.Dispose()
          End If
```

```
        End Try
      End If
    End With
  Catch ex As Exception
    Ok = False
    errorMessage = ex.Message
  End Try

  ' Send Request and get Response
  Try
    Response = DirectCast(Request.GetResponse(), System.Net.HttpWebResponse)
  Catch ex As System.Net.WebException
    ' Not all exceptions should be treated as errors
    If ex.Response Is Nothing OrElse Not (TypeOf ex.Response Is System.Net.HttpWebResponse) Then
      Ok = False
      errorMessage = ex.Message
    Else
      ' Valid Response received
      response = DirectCast(ex.Response, System.Net.HttpWebResponse)
    End If
  Catch ex As Exception
    Ok = False
    errorMessage = ex.Message
  End Try

  ' Get Response details
  If Ok AndAlso response IsNot Nothing Then
    ' Response (as both a Byte array and Text to cater for both situations)
    If response.ContentLength = 0 Then
      ' No Response Content
      responseText = ""
      ReDim responseBytes(0)
    Else
      ' Get either Text or Binary Response
      If response.ContentType.StartsWith("text", StringComparison.OrdinalIgnoreCase) OrElse
InStr(response.ContentType, "/json", CompareMethod.Text) <> 0 Then
        ' Text
        srText = New System.IO.StreamReader(response.GetResponseStream())
        responseText = srText.ReadToEnd()
        srText.Close()
      Else
        ' Binary
        MemoryStream = New System.IO.MemoryStream()
        Stream = response.GetResponseStream()
        Do
          Bytes = Stream.Read(Buffer, 0, 1023)
          If Bytes > 0 Then MemoryStream.Write(Buffer, 0, Bytes)
        Loop While Bytes > 0
        responseBytes = MemoryStream.ToArray()
      End If
    End If

    ' Status Code
    statusCode = CInt(response.StatusCode)
  End If

  Return Ok

End Function

Public Function HashSha256(value As String,
                           key As String,
                           ByRef salt As String) As String

  Dim HashBytes As Byte()
  Dim i As Integer
  Dim rnd As Random
  Dim SaltValueKeyBytes As Byte()
  Dim Sha256 As System.Security.Cryptography.SHA256

  ' Generate Salt (could just as easily be passed in but this will generate a random one regardless)
  rnd = New Random(CInt(Now.Ticks Mod Int32.MaxValue))
  salt = ""
  For i = 1 To 10
    salt &= Chr(65 + rnd.Next(0, 25))
  Next

  ' Convert Salt+Value+Key into a byte array
  SaltValueKeyBytes = System.Text.Encoding.UTF8.GetBytes(salt & value & key)

  ' Hash
```

```
    Sha256 = New System.Security.Cryptography.SHA256Managed()
    HashBytes = Sha256.ComputeHash(SaltValueKeyBytes)

    ' Base-64 encode
    Return System.Convert.ToBase64String(HashBytes)

End Function
```

Authentication Response

Each of the Authentication methods return a Session Details response, which includes a Session Token and other details, e.g.

```
<SessionDetails xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SessionToken>d8f86085J7Bts5lnWyHVLeEkQ30Dug==:LAyh9/NQ2UZ/FAg8qB1AN18hT+OQWC6YY36IvqMreZ8=</SessionToken>
  <RoleType>User</RoleType>
  <FullName>Administrator</FullName>
</SessionDetails>
```

```
<SessionDetails xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SessionToken>84506f17FarOJkJL0erGpKbIgonNbg==:Y9rCcvGiIh8nGwwFil5IIEZJi6sewsNGCOsGWceefp/2Q8FmGX6tyegLMDLEinO5</SessionToken>
  <RoleType>Client</RoleType>
  <FullName>John Smith</FullName>
  <ClientId>C10000</ClientId>
  <ClientLastLoginDate xsi:nil="true" />
  <ClientBranchId>M</ClientBranchId>
  <ClientBranchName>Head Office</ClientBranchName>
  <ClientBranchAddressPhysicalFormatted>31 Auckland Road Napier</ClientBranchAddressPhysicalFormatted>
  <ClientBranchAddressPostalFormatted>PO BOX 31 Napier</ClientBranchAddressPostalFormatted>
  <ClientBranchContactMethodPhone>(06) 8355 2668</ClientBranchContactMethodPhone>
  <ClientBranchContactMethodEmail>ho@testcompany.xyz</ClientBranchContactMethodEmail>
</SessionDetails>
```

Supplying the Session Token with a Request

The Session Token returned when authenticating (signing in) must be supplied with all authenticated requests (i.e., just about anything other than Ping).

In a typical Web application, the Session Token might be stored in Session state.

The **Authentication Code Sample** above demonstrated issuing an authenticated request to retrieve a list of a Client's Accounts. The authenticated request included a special HTTP Authorization header, e.g.:

```
GET http://localhost/finPOWERConnectWS2/Api/Client/GetAccounts?clientId=C10000
Content-Type: text/xml
Authorization: AuthFinWs token="0fb1121adfPi6BcHo48oIgTWMo9+ZA==:rW5lqC5"
```

NOTE: The **token=** portion of the header specifies the Session Token (this has been truncated for clarity in the above example).

When to Re-Authenticate

As mentioned above, authenticating (signing in), returns a Session Token that must be sent with any request that requires authentication.

The Session Token is valid for 24 hours, after which, supplying it with a request will fail.

Therefore, you may wish to consider the following scenarios and solutions:

- Never re-authenticate
  - When the User (or Client) first signs in, store the Session Token in Session State. Typically, for financial applications, Session State on a Web Server is set to a short period, e.g., 20 minutes of inactivity.

- Therefore, if you are sure that a User is never going to be continuously connected to your Web application for more than 24 hours, this approach is suitable.
- This is the method the Client Connect sample uses.

- Re-authenticate a User/ Client after a set period
  - If a User (or Client) is likely to remain signed in to your Web application for more than 24 hours, you may wish to re-authenticate after a set period, e.g., every 12 hours.
  - When the User (or Client) first signs in, store the Session Token together with the authentication time and the User (or Client's) Id and Password in **Session State**.
    - **NOTE:** If using out-of-process Session State, you may wish to encrypt this information as a precaution.
  - Prior to performing a Web Service request, check if the Session Token is more than 12 hours old and, if so, re-authenticate using the Id and Password stored in Session State. Update the Session Token and authentication time held in Session State.
- Ad-hoc access, i.e., no User or Client is signed in
  - If your Web application has no concept of a User (or Client) signing in, e.g., an application that provides a loan application form on a public Website, you will need to store the credentials of a finPOWER Connect User somewhere in your application. You will then need to do one of the following:
    - Re-authenticate prior to performing a Web Service request. This provides a new Session Token that you never need to store.
      - **NOTE:** This approach may bloat the finPOWER Connect audit log.
    - When first performing a Web Service request, authenticate and store the Session Token together with the authentication time in **Application State**.
      - Prior to performing a Web Service request, check if the Session Token is more than 12 hours old and, if so, re-authenticate and update the Session Token and authentication time held in Application State.

Token-Based User Authentication

As of finPOWER Connect 2.03.02, token-based authentication is supported.

This allows an external Web Application, if launched from within the finPOWER Connect Windows interface, to be passed a special token for the currently logged in User. This token can then be used by the external Web Application to authenticate (sign-in) without the User having to re-enter their credentials.

Token-based authentication is implemented within finPOWER Connect via "WebServicesToken" type Application Shortcuts.

E.g., if a Summary Page in finPOWER Connect contains a link with the following URL:

```
WebServicesToken?url=https%3A%2F%2Fwww.yourwebapplication.com%2Ftest.aspx%3Fwst%3D%5Btoken%5D&
openInternal=true&title=Your Web Application
```

When the User clicks the URL, finPOWER Connect will create a special token.

The URL parameter of the Application Shortcut which is:

```
https://www.yourwebapplication.com/test.aspx?wst=[token]
```

Will be updated to include the token, e.g.:

```
https://www.yourwebapplication.com/test.aspx?wst=
6JVAwQ0e7fGFEg3alxQ5xobSSqUym%2FS54AKr%2Fi%2F2YEId3qh3sQYkY%2Bstyc2VkBzM
```

And the URL followed.

The external Web Application should then do the following in the test.aspx page:

- Get the token from the URL (in this case, from the parameter named "wst").
- Use the Web Subscriber's Secret Key to create an SHA256 hash (and hash salt) of this token.
  - **NOTE:** A sample SHA256 hash function is included in the code sample earlier in this secton.
- Call the Authentication/**AuthenticateUserToken** Web Service passing in the relevant parameters.

NOTE: By default, token-based logins are disabled in finPOWER Connect. They can be enabled from Tools, Global Settings, Web, General page.

A token expiry period can also be specified from this page. This defines how long the token generated by finPOWER Connect is valid for. By default this is set to 60 seconds.

Dates

Web Service dates use the ISO 8601 standard regardless of whether the date is passed as part of a request URL or is included in an XML or JSON response.

Web Services return dates in UTC (Coordinated Universal Time) regardless of how they are stored within the finPOWER Connect database or displayed in the finPOWER Connect Windows interface.

WARNING: Any dates provided as parameters to Web Services will be parsed according to the time zone on the Web Server. Therefore, it is advisable to always provide dates in UTC format.

Dates are covered in detail in the **Dates** topic of the Web Services API reference.

Javacript

In Javascript, when deserialising a JSON object, dates are treated as strings and are not automatically converted to dates. This is explained fully in this article by Rick Strahl:

http://weblog.west-wind.com/posts/2014/Jan/06/JavaScript-JSON-Date-Parsing-and-real-Dates

Serialising and Deserialising XML

Some programming languages support serialisation and deserialisation of XML from and to an object.

In many cases, this can make parsing a response (or forming a request) easier than dealing directly with XML.

NOTE: This section is intended for external applications consuming the Web Services. For Custom Web Service Scripts, see the **finPOWER Connect Custom Web Services Programming Guide** document.

This section gives code samples of both serialisation and deserialisation. None of these samples use the finPOWER Connect business layer.

Serialisation/ Deserialisation can be used with complex objects and collections (E.g., objects that contain other objects). A Google search will provide many examples of how to achieve this.

Serialisation

Serialisation is the process of taking an object and automatically creating an XML String that represents that object.

The following examples will all serialise the simple ClientDetails object detailed below into XML:

```
ClientDetails
  ClientId (String)
  FirstName (String)
  LastName (String)
  DateOfBirth (Date)
```

Each of the code samples contains a **`SerialiseObjectToXmlString`** function which returns a `Boolean` value based upon whether it was successful or not. If unsuccessful, an error message is returned instead of the XML.

Visual Basic

```
Public Sub Main()

  Dim ClientDetails As clsClientDetails
  Dim Xml As String

  ' Create Object
  ClientDetails = New clsClientDetails()
  With ClientDetails
    .ClientId = "C10000"
    .FirstName = "John"
    .LastName = "Smith"
    .DateOfBirth = New Date(1970, 9, 5)
  End With

  ' Serialise
  If SerialiseObjectToXmlString(ClientDetails, Xml) Then
    MsgBox(Xml)
  Else
    MsgBox(Xml, MsgBoxStyle.Exclamation)
  End If

End Sub

Public Function SerialiseObjectToXmlString(obj As Object, ByRef xml As String) As Boolean

  Dim ms As System.IO.MemoryStream
  Dim Ok As Boolean
  Dim XmlSerializer As System.Xml.Serialization.XmlSerializer

  ' Assume Success
  Ok = True

  ' Initialise ByRef Parameters
  xml = ""

  ' Serialise
  Try
    XmlSerializer = New System.Xml.Serialization.XmlSerializer(obj.GetType())
    ms = New System.IO.MemoryStream()
    XmlSerializer.Serialize(ms, obj)
    xml = System.Text.Encoding.UTF8.GetString(ms.GetBuffer(), 0, CInt(ms.Length))
    ms.Close()
  Catch ex As Exception
    ' Failed (return error message instead of XML)
    Ok = False
    xml = ex.Message
  Finally
    If ms IsNot Nothing Then ms.Dispose()
  End Try

  Return Ok

End Function

<System.Xml.Serialization.XmlType("ClientDetails")>
Public Class clsClientDetails
  Public ClientId As String
  Public FirstName As String
```

```
  Public LastName As String
  Public DateOfBirth As Date
End Class
```

C#

```
public void Main()
{
  clsClientDetails ClientDetails;
  string Xml = "";

  // Create Object
  ClientDetails = new clsClientDetails {
    ClientId = "C10000",
    FirstName = "John",
    LastName = "Smith",
    DateOfBirth = new DateTime(1970, 9, 5)
  };

  // Serialise
  if(SerialiseObjectToXmlString(ClientDetails, ref Xml)) {
    System.Windows.Forms.MessageBox.Show(Xml);
  }
  else {
    System.Windows.Forms.MessageBox.Show(Xml, "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
  }
}

public bool SerialiseObjectToXmlString(Object obj, ref string xml)
{
  System.IO.MemoryStream ms = null;
  bool Ok;
  System.Xml.Serialization.XmlSerializer XmlSerializer;

  // Assume Success
  Ok = true;

  // Initialise ByRef Parameters
  xml = "";

  // Serialise
  try {
    XmlSerializer = new System.Xml.Serialization.XmlSerializer(obj.GetType());
    ms = new System.IO.MemoryStream();
    XmlSerializer.Serialize(ms, obj);
    xml = System.Text.Encoding.UTF8.GetString(ms.GetBuffer(), 0, (int)ms.Length);
    ms.Close();
  }
  catch(Exception ex) {
    // Failed (return error message instead of XML)
    Ok = false;
    xml = ex.Message;
  }
  finally {
    if(ms != null) ms.Dispose();
  }

  return Ok;
}

[System.Xml.Serialization.XmlType("ClientDetails")]
public class clsClientDetails {
  public string ClientId;
  public string FirstName;
  public string LastName;
  public DateTime DateOfBirth;
}
```

Deserialisation

Deserialisation is the process of taking an XML String and automatically creating and populating an object from the content of the XML.

The following examples will deserialise the following XML into the simple ClientDetails object used in the previous section.

```
ClientDetails
  ClientId (String)
  FirstName (String)
  LastName (String)
  DateOfBirth (Date)
```

Each of the code samples contains a **`DeserialiseObjectFromXmlString`** function which returns a `Boolean` value based upon whether it was successful or not. If unsuccessful, an error message is returned instead of the deserialised object.

Visual Basic

```
Public Sub Main()

  Dim ClientDetails As clsClientDetails
  Dim tempObject As Object
  Dim Xml As String

  ' Define XML
  Xml &= "<ClientDetails>"
  Xml &= "<ClientId>C10000</ClientId>"
  Xml &= "<FirstName>John</FirstName>"
  Xml &= "<LastName>Smith</LastName>"
  Xml &= "<DateOfBirth>1970-09-05T00:00:00</DateOfBirth>"
  Xml &= "</ClientDetails>"

  ' Deserialise into Object
  If DeserialiseXmlStringToObject(Xml, GetType(clsClientDetails), tempObject) Then
    ClientDetails = DirectCast(tempObject, clsClientDetails)
    MsgBox("Deserialised ClientId=" & ClientDetails.ClientId)
  Else
    MsgBox(CStr(tempObject), MsgBoxStyle.Exclamation)
  End If

End Sub

Public Function DeserialiseXmlStringToObject(xml As String,
                                             objType As Type,
                                             ByRef obj As Object) As Boolean

  Dim ms As System.IO.MemoryStream
  Dim Ok As Boolean
  Dim XmlSerializer As System.Xml.Serialization.XmlSerializer

  ' Assume Success
  Ok = True

  ' Initialise ByRef Parameters
  obj = Nothing

  ' Deserialise
  Try
    ' Create Serialiser
    XmlSerializer = New System.Xml.Serialization.XmlSerializer(objType)

    ' Deserialise
    ms = New System.IO.MemoryStream(System.Text.Encoding.UTF8.GetBytes(xml))
    obj = XmlSerializer.Deserialize(ms)
    ms.Close()
  Catch ex As Exception
    Ok = False
    obj = ex.Message
  Finally
    If ms IsNot Nothing Then ms.Dispose()
  End Try
```

```
  Return Ok

End Function

<System.Xml.Serialization.XmlType("ClientDetails")>
Public Class clsClientDetails
  Public ClientId As String
  Public FirstName As String
  Public LastName As String
  Public DateOfBirth As Date
End Class
```

C#

```
public void Main(object sender, EventArgs e)
{
  clsClientDetails ClientDetails;
  Object tempObject = null;
  string Xml;

  // Define XML
  Xml = "";
  Xml += "<ClientDetails>";
  Xml += "<ClientId>C10000</ClientId>";
  Xml += "<FirstName>John</FirstName>";
  Xml += "<LastName>Smith</LastName>";
  Xml += "<DateOfBirth>1970-09-05T00:00:00</DateOfBirth>";
  Xml += "</ClientDetails>";

  // Deserialise into Object
  if(DeserialiseXmlStringToObject(Xml, typeof(clsClientDetails), ref tempObject))
  {
    ClientDetails = (clsClientDetails)tempObject;
    System.Windows.Forms.MessageBox.Show("Deserialised ClientId=" + ClientDetails.ClientId);
  }
  else
  {
    System.Windows.Forms.MessageBox.Show((string)tempObject, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
  }
}

bool DeserialiseXmlStringToObject(string xml, Type objType, ref Object obj)
{
  System.IO.MemoryStream ms = null;
  bool Ok;
  System.Xml.Serialization.XmlSerializer XmlSerializer;

  // Assume Success
  Ok = true;

  // Initialise ByRef Parameters
  obj = null;

  // Deserialise
  try
  {
    // Create Serialiser
    XmlSerializer = new System.Xml.Serialization.XmlSerializer(objType);

    // Deserialise
    ms = new System.IO.MemoryStream(System.Text.Encoding.UTF8.GetBytes(xml));
    obj = XmlSerializer.Deserialize(ms);
    ms.Close();
  }
  catch (Exception ex)
  {
    Ok = false;
    obj = ex.Message;
  }
  finally
  {
    if (ms != null) ms.Dispose();
  }

  return Ok;
}

[System.Xml.Serialization.XmlType("ClientDetails")]
```

```
public class clsClientDetails {
  public string ClientId;
  public string FirstName;
  public string LastName;
  public DateTime DateOfBirth;
}
```

Serialising and Deserialising JSON

Some programming languages support serialisation and deserialisation of JSON (JavaScript Object Notation) from and to an object. The most obvious is client-side JavaScript which contains the `JSON.stringify` and `JSON.parse` methods to handle this.

This section is aimed more towards server-side serialisation and deserialisation.

NOTE: This section is intended for external applications consuming the Web Services. For Custom Web Service Scripts, see the **finPOWER Connect Custom Web Services Programming Guide** document.

This section gives code samples of both serialisation and deserialisation. None of these samples use the finPOWER Connect business layer.

Serialisation/ Deserialisation can be used with complex objects and collections (e.g., objects that contain other objects). A Google search will provide many examples of how to achieve this.

Serialisation

Serialisation is the process of taking an object and automatically creating a JSON String that represents that object.

The following example serialises the simple ClientDetails object detailed below into JSON:

```
ClientDetails
  ClientId (String)
  FirstName (String)
  LastName (String)
  DateOfBirth (Date)
```

The code samples (apart from the JavaScript sample) contain a **`SerialiseObjectToJsonString`** function which returns a `Boolean` value based upon whether it was successful or not. If unsuccessful, an error message is returned instead of the JSON object.

JavaScript

This client-side example relies on no external libraries since JSON serialisation is supported by all modern Web browsers (and Internet Explorer as of version 8).

```
// Create JSON object
var o = {};
o.ClientId = "C10000";
o.FirstName = "John";
o.LastName = "Smith";
o.DateOfBirth = new Date(1970, 9, 23);

// Serialise
var jsonText = JSON.stringify();
```

Visual Basic

DataContractJsonSerializer

This first example uses the .NET `DataContractJsonSerializer` class which is available in the `System.Runtime.Serialization.Json` assembly.

```
Imports System.Runtime.Serialization.Json

Public Sub Main()

  Dim ClientDetails As clsClientDetails
  Dim JsonText As String

  ' Create Object
  ClientDetails = New clsClientDetails()
  With ClientDetails
    .ClientId = "C10000"
    .FirstName = "John"
    .LastName = "Smith"
    .DateOfBirth = New Date(1970, 9, 23)
  End With

  ' Serialise
  If SerialiseObjectToJsonString(ClientDetails, JsonText) Then
    MsgBox(JsonText)
  Else
    MsgBox(JsonText, MsgBoxStyle.Exclamation)
  End If

End Sub

Public Function SerialiseObjectToJsonString(obj As Object, ByRef jsonText As String) As Boolean

  Dim Ok As Boolean
  Dim ms As System.IO.MemoryStream
  Dim Serialiser As DataContractJsonSerializer
  Dim SerialiserSettings As DataContractJsonSerializerSettings
```

```
  ' Assume Success
  Ok = True

  ' Initialise ByRef Parameters
  jsonText = ""

  ' Serialise
  Try
    ' Serialise
    ' NOTE: Ensure that the standard date format is used
    ms = New System.IO.MemoryStream()
    SerialiserSettings = New DataContractJsonSerializerSettings()
    SerialiserSettings.DateTimeFormat = New Runtime.Serialization.DateTimeFormat("yyyy-MM-
dd'T'HH:mm:ssZ")
    Serialiser = New DataContractJsonSerializer(obj.GetType(), SerialiserSettings)
    Serialiser.WriteObject(ms, obj)

    ' Get JSON text
    jsonText = System.Text.Encoding.Default.GetString(ms.ToArray())
  Catch ex As Exception
    ' Failed (return error message instead of XML)
    Ok = False
    jsonText = ex.Message
  Finally
    If ms IsNot Nothing Then ms.Dispose()
  End Try

  Return Ok

End Function

Public Class clsClientDetails
  Public ClientId As String
  Public FirstName As String
  Public LastName As String
  Public DateOfBirth As Date
End Class
```

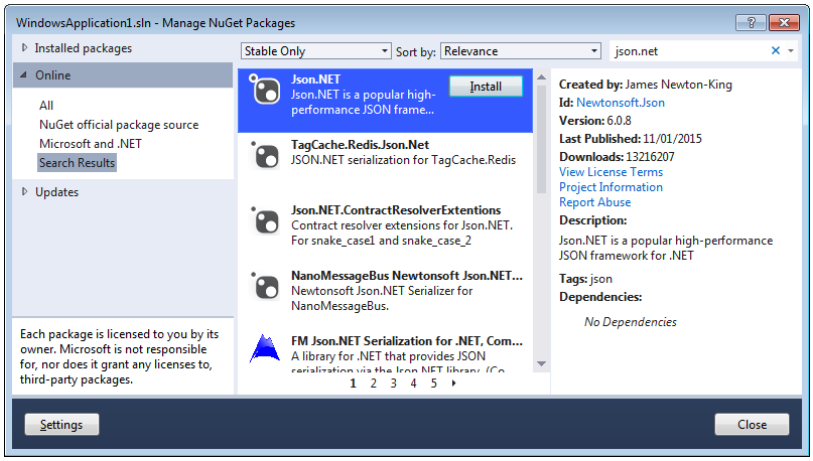
NOTE: By default, the `DataContractJsonSerializer` does not handle dates well hence we must pass it a `DataContractJsonSerializerSettings` object which is only available in .NET 4.5 and later.

Newtonsoft.Json.NET Library

This next example uses the `Newtonsoft.Json` library which is preferred by many developers and has become the standard in many of Microsoft's own Web examples.

To add the Newtonsoft library to a Visual Studio 2012 project, do the following:

- From the **Tools** menu, select **Library Package Manager**, **Manage NuGet Packages for Solution**
- Enter **json.net** into the Search box in the top-right of the Manage NuGet Packages form
- Select the **Json.NET** entry:



- Click the **Install** button.

The above steps will install the Newtonsoft Json.NET library and add references and a packages.config file to the Visual Studio project.

```
Imports Newtonsoft.Json

Public Sub Main()

  Dim ClientDetails As clsClientDetails
  Dim JsonText As String

  ' Create Object
  ClientDetails = New clsClientDetails()
  With ClientDetails
    .ClientId = "C10000"
    .FirstName = "John"
    .LastName = "Smith"
    .DateOfBirth = New Date(1970, 9, 23)
  End With

  ' Serialise
  If SerialiseObjectToJsonString(ClientDetails, JsonText) Then
    MsgBox(JsonText)
  Else
    MsgBox(JsonText, MsgBoxStyle.Exclamation)
  End If

End Sub

Public Function SerialiseObjectToJsonString(obj As Object, ByRef jsonText As String) As Boolean

  Dim Ok As Boolean

  ' Assume Success
  Ok = True

  ' Initialise ByRef Parameters
  jsonText = ""

  ' Serialise
  Try
    ' Serialise (ensure Enums as serialised as Strings rather than Integers)
    jsonText = Newtonsoft.Json.JsonConvert.SerializeObject(obj,
                                                           New
Newtonsoft.Json.Converters.StringEnumConverter())
  Catch ex As Exception
    ' Failed (return error message instead of XML)
    Ok = False
    jsonText = ex.Message
  End Try

  Return Ok

End Function

Public Class clsClientDetails
  Public ClientId As String
  Public FirstName As String
  Public LastName As String
```

```
    Public DateOfBirth As Date
End Class
```

NOTE: The above example configures the Newtonsoft serialiser to format Enums as Strings rather than Integers.

NOTE: The Newtonsoft sample is much simpler than the `DataContractJsonSerializer` and also handles dates correctly without any special configuration.

Deserialisation

Deserialisation is the process of taking a JSON String and automatically creating and populating an object from the content of the JSON.

The following examples will deserialise a JSON String into the simple ClientDetails object used in the previous section. The exception to this is the JavaScript example which simply deserialises into a JavaScript object.

```
ClientDetails
  ClientId (String)
  FirstName (String)
  LastName (String)
  DateOfBirth (Date)
```

Each of the code samples contains a **`DeserialiseObjectFromJsonString`** function which returns a `Boolean` value based upon whether it was successful or not. If unsuccessful, an error message is returned instead of the deserialised object.

JavaScript

This sample relies on no external libraries since JSON serialisation is supported by all modern Web browsers (and Internet Explorer as of version 8).

```
var o = JSON.parse(jsonText);

window.alert(o.FirstName);
```

Visual Basic

DataContractJsonSerializer

This first example uses the .NET `DataContractJsonSerializer` class which is available in the `System.Runtime.Serialization.Json` assembly.

```
Imports System.Runtime.Serialization.Json

Public Sub Main()

  Dim ClientDetails As clsClientDetails
  Dim tempObject As Object
  Dim JsonText As String

  ' Define JSON
  JsonText = ""
  JsonText &= "{'ClientId':'C10000',"
  JsonText &= "'FirstName':'John',"
  JsonText &= "'LastName':'Smith',"
  JsonText &= "'DateOfBirth':'1970-09-23T00:00:00Z'}"
  JsonText = Replace(JsonText, "'", Chr(34))

  ' Deserialise into Object
  If DeserialiseJsonStringToObject(JsonText, GetType(clsClientDetails), tempObject) Then
    ClientDetails = DirectCast(tempObject, clsClientDetails)
    MsgBox("Deserialised ClientId=" & ClientDetails.ClientId)
  Else
    MsgBox(CStr(tempObject), MsgBoxStyle.Exclamation)
  End If

End Sub

Public Function DeserialiseJsonStringToObject(jsonText As String,
                                              objType As Type,
                                              ByRef obj As Object) As Boolean

  Dim Bytes As Byte()
  Dim ms As System.IO.MemoryStream
  Dim Ok As Boolean
  Dim Serialiser As DataContractJsonSerializer
  Dim SerialiserSettings As DataContractJsonSerializerSettings
```

```
  ' Assume Success
  Ok = True

  ' Initialise ByRef Parameters
  obj = Nothing

  ' Deserialise
  Try
    ' Get JSON text as byte array
    Bytes = System.Text.Encoding.ASCII.GetBytes(jsonText)
    ms = New System.IO.MemoryStream(Bytes)

    ' Deserialise
    ' NOTE: Ensure that the standard date format can be deserialised
    SerialiserSettings = New DataContractJsonSerializerSettings()
    SerialiserSettings.DateTimeFormat = New Runtime.Serialization.DateTimeFormat("yyyy-MM-
dd'T'HH:mm:ssZ")
    Serialiser = New DataContractJsonSerializer(objType, SerialiserSettings)
    obj = Serialiser.ReadObject(ms)
  Catch ex As Exception
    Ok = False
    obj = ex.Message
  Finally
    If ms IsNot Nothing Then ms.Dispose()
  End Try

  Return Ok

End Function

Public Class clsClientDetails
  Public ClientId As String
  Public FirstName As String
  Public LastName As String
  Public DateOfBirth As Date
End Class
```

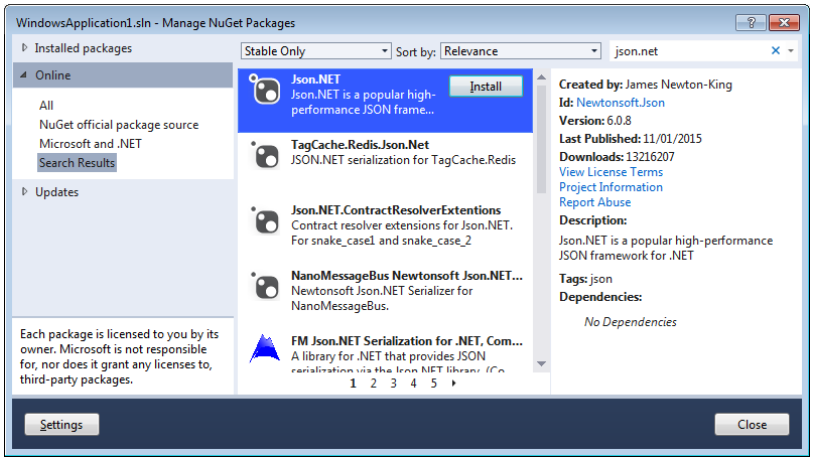
NOTE: By default, the `DataContractJsonSerializer` does not handle dates well hence we must pass it a `DataContractJsonSerializerSettings` object which is only available in .NET 4.5 and later.

Newtonsoft.Json.NET Library

This next example uses the `Newtonsoft.Json` library which is preferred by many developers and has become the standard in many of Microsoft's own Web examples.

To add the Newtonsoft library to a Visual Studio 2012 project, do the following:

- From the **Tools** menu, select **Library Package Manager**, **Manage NuGet Packages for Solution**
- Enter **json.net** into the Search box in the top-righht of the Manage NuGet Packages form
- Select the **Json.NET** entry:



- Click the **Install** button.

The above steps will install the Newtonsoft Json.NET library and add references and a packages.config file to the Visual Studio project.

```
Imports Newtonsoft.Json

Public Sub Main()

  Dim ClientDetails As clsClientDetails
  Dim tempObject As Object
  Dim JsonText As String

  ' Define JSON
  JsonText = ""
  JsonText &= "{'ClientId':'C10000',"
  JsonText &= "'FirstName':'John',"
  JsonText &= "'LastName':'Smith',"
  JsonText &= "'DateOfBirth':'1970-09-23T00:00:00Z'}"
  JsonText = Replace(JsonText, "'", Chr(34))

  ' Deserialise into Object
  If DeserialiseJsonStringToObject(JsonText, GetType(clsClientDetails), tempObject) Then
    ClientDetails = DirectCast(tempObject, clsClientDetails)
    MsgBox("Deserialised ClientId=" & ClientDetails.ClientId)
  Else
    MsgBox(CStr(tempObject), MsgBoxStyle.Exclamation)
  End If

End Sub

Public Function DeserialiseJsonStringToObject(jsonText As String,
                                              objType As Type,
                                              ByRef obj As Object) As Boolean

  Dim Ok As Boolean
  Dim jsonObject As Newtonsoft.Json.Linq.JObject

  ' Assume Success
  Ok = True

  ' Initialise ByRef Parameters
  obj = Nothing

  ' Deserialise
  Try
    jsonObject = Newtonsoft.Json.Linq.JObject.Parse(jsonText)
    obj = jsonObject.ToObject(objType)
  Catch ex As Exception
    Ok = False
    obj = ex.Message
  End Try

  Return Ok

End Function

Public Class clsClientDetails
  Public ClientId As String
  Public FirstName As String
```

```
    Public LastName As String
    Public DateOfBirth As Date
End Class
```

NOTE: The Newtonsoft sample is much simpler than the `DataContractJsonSerializer` and also handles dates correctly without any special configuration.

Visual Basic Code Examples

Visual Basic code examples are included in the **/Samples** folder of the Web Services. All of these samples are ASP.NET Web Forms and therefore contain an **.aspx** and an **.aspx.vb** file.

Each sample is included in a sub-folder which contains all of the required files.

The following table lists these samples:

Sample	Description	Other Details
ConnectUser1VB	Connect as a finPOWER Connect User and retrieve a list of Client Accounts.	
CustomLoanQuote1VB	Enter simple Loan and Client details and use a custom Web Service to create a Quote Account and a Client.	Script CustomLoanQuote1VB.xml must be imported into finPOWER Connect.
LoanApplication1VB	Full 'payday' type Loan Application example.	Script LoanApplication1VB.xml must be imported into finPOWER Connect.

NOTE: As of version 2.02.06, these samples can be run directly from the /Samples folder of the Web Services installation (although you may need to modify the constants at the top of each sample page as detailed in the next section).

Creating a new Visual Studio Project

The following steps detail how to get the ConnectUser1VB sample running in a new Visual Studio project.

- Create a new empty Web Application in Visual Studio 2012 (or Visual Studio Express 2012 for Web).
- Copy the entire contents of the Web Services **/Samples/ConnectUser1VB** folder into the new Web Application.
  - Include all files (except for any .xml files) in the Web Application project.
- Ensure you have a Web Subscriber record set up in finPOWER Connect.
  - In finPOWER Connect, open the database that the Web Services are connected to.
  - From the **Tools** menu, select **Web** and then **Web Subscribers**.
  - Add a new Web Subscriber record with the following details:
    - Code: TEST
    - Description: Test Web Subscriber
- View the .vb portion of all sample pages and update the following constants to allow connection to the Web Services:
  - WEB_SERVICES_URL
  - WEB_SUBSCRIBER_ID
  - WEB_SUBSCRIBER_SECRET_KEY

- Run the Web Application and ensure that the **ConnectUser1VB.aspx** page runs correctly.
  - This is the simplest example, therefore ensure this runs correctly before attempting to run any other examples.
- Some samples may require a custom Web Service Script.
  - E.g., the CustomLoanQuote1VB sample requires that the corresponding .xml file is imported into the finPOWER Connect Scripts library as follows:
    - In finPOWER Connect, open the database that the Web Services are connected to.
    - From the **Admin** menu, select **Scripts**.
    - Use the **Import** action to import the .xml file.
    - Save the Script.

Loan Application 1 (Visual Basic)

This sample is included in the /Samples/LoanApplication1VB folder.

Overview

This is a fully self-contained example of using Web Services to provide a 'payday' type Loan Application page.

Files

The sample consists of the following files:

- Logo.png
  - The logo image used in the sample.
- Styles.css
  - The Cascading Stylesheet file used in the sample.
- LoanApplication1VB.xml
  - The Custom Web Services used by the sample.
- LoanApplication1VB.aspx
  - The HTML and corresponding code-behind ASP.NET code.

Configuration

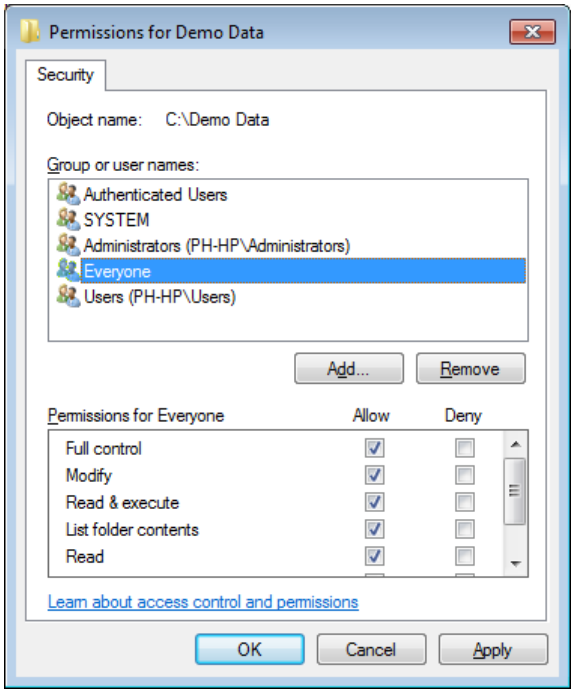
This sample is designed to be able to run against the demonstration database included with a finPOWER Connect installation (finDemo_NZ.mdb).

This database contains the following:

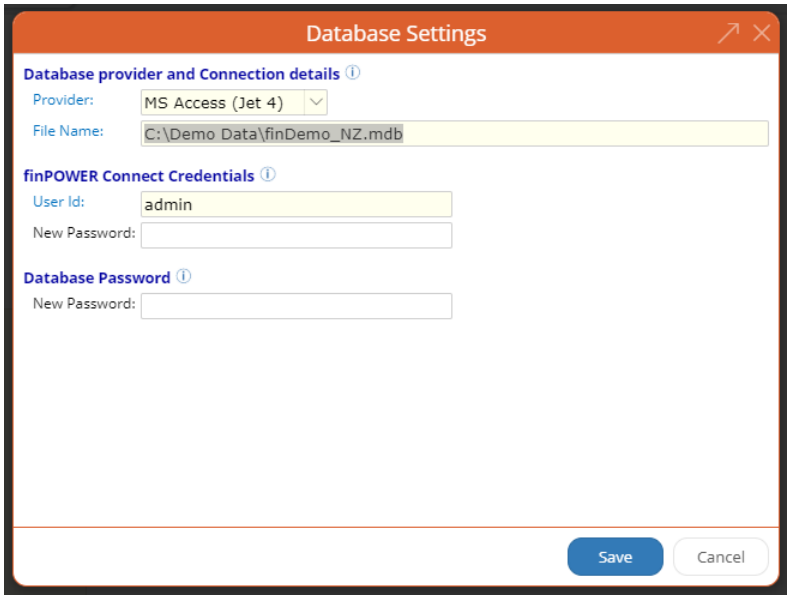
- A 'DEMO' Web Subscriber record
  - This has a Secret Key of ' 11111111111111111111111111111111' which matches the constant at the top of all Web Services samples.
- A 'DEMO' Account Application Type record.
- A 'PDL' Account Type record.
- Other records such as Client Types and Contact Methods required by this sample.

To configure the sample to run from within the Web Services installation (as opposed to creating a new Visual Studio project):

- Create a C:\Demo Data folder on the Web Server hosting the Web Services.
  - For the sake of a simple example (and not something that should be done in a production environment), set the permissions on this folder to allow 'Everyone' full access:



- Sign in to the Web Services Administration facility.
  - Configure the Web Services to point to this database:



  - And the finPOWER Credentials to connect as:
    - User Id: **admin**
    - Password: **admin**
- Open the demonstration database (C:\Demo Data\finDemo_NZ.mdb) in finPOWER Connect.
  - Sign in as the same credentials as shown above, i.e.:
    - User Id: **admin**
    - Password: **admin**
- Import the Custom Web Service Script:
  - Open the Scripts form (Admin, Scripts).
  - Click the 'Import' action under the 'Utilities' heading on the left of the form.
  - Locate the LoanApplication1VB.xml file and import it.

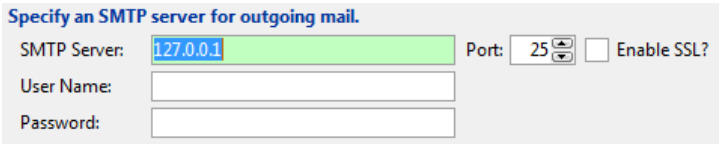
By default, constants at the top of the Script determine the functionality of the sample, e.g.:

- `CreateAccountApp`
  - Determines whether an Account (the default) or an Account Application will be created.
- VERIFICATION_METHOD
  - Determines whether a verification Email or SMS should be sent to the Applicant as part of the application process.
    - Set to a blank String to disable verification.
      - You will also need to modify the corresponding constant in the LoanApplication1VB.aspx file.
- CONFIRMATION_METHOD
  - Determines whether a configuration Email or SMS should be sent to the Applicant upon completing the application.
    - Set to a blank String to disable confirmation.
    - You will also need to modify the corresponding constant in the LoanApplication1VB.aspx file.

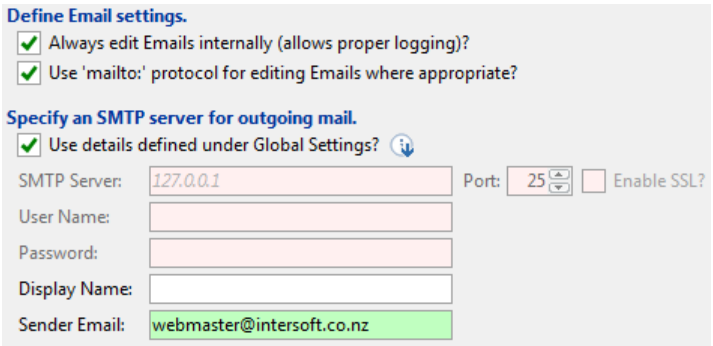
If you are using Verification or Confirmation Emails and SMS's, you will need to ensure that the finPOWER Connect database is configured to allow this.

Typically, a Web Configuration (Tools, Web, Web Configurations) would be used but, for this example, it is simpler to use Global Settings and User Preferences:

- Open the Global Settings form (Tools, Global Settings).
  - Select the Messaging, Email page.
  - Enter details of an SMTP server that is available from the Web Server, e.g.:



- Open the User Preferences form (Tools, User Preferences).
  - Select the Messaging, Email page.
  - Enter details of the Email Address under which to send the confirmation Email, e.g.:



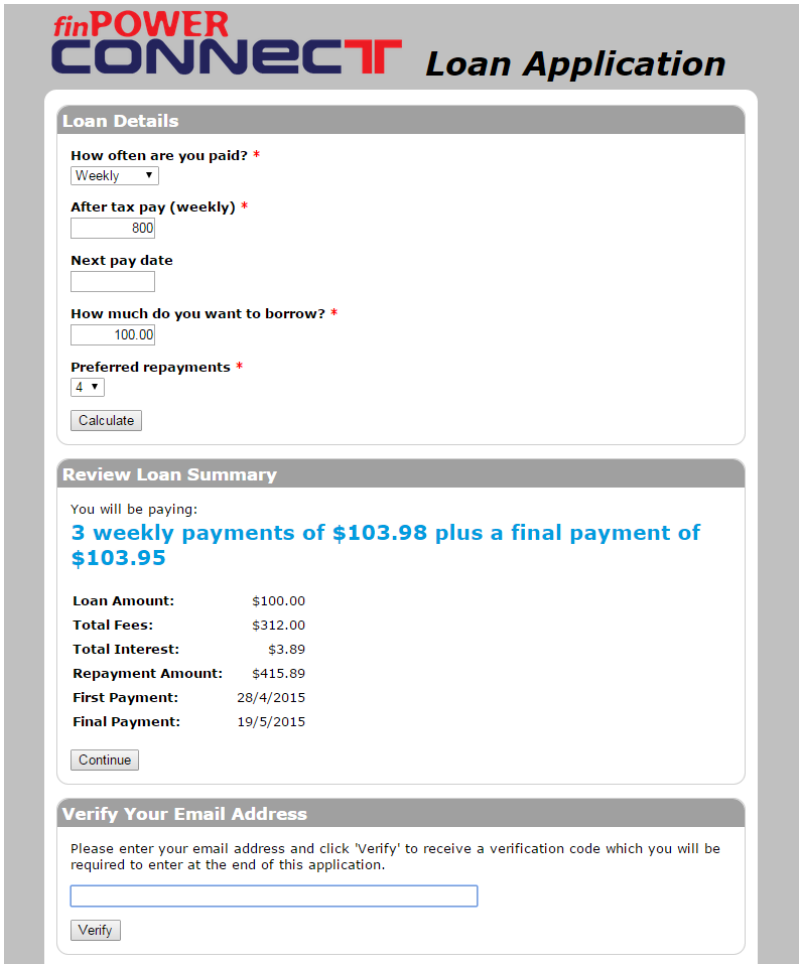
NOTE: If using SMS for verification or confirmation, repeat the above steps but use the Messaging, SMS pages in Global Settings and User Preferences to configure an SMS provider.

Running the Sample

Using a Web browser, navigate to the sample, e.g.:

`http://localhost/finPOWERConnectWs2/Samples/LoanApplication1VB/LoanApplication1VB.aspx`

The sample provides a single page form with expandable sections into which loan application information can be entered, e.g.:



NOTE: The sample will create either a 'Quote' Account and associated Client or an Account Application depending on the constants configured at the top of the Custom Web Service in finPOWER Connect.

Logo

The included logo file is the finPOWER Connect logo:

This file is 700 x 130 pixels.

However, code within the HTML page (LoanApplication1VB.aspx) sets the logo to scale to 50% (i.e., 350 x 65 pixels):

```
<!-- Scale logo to 50% native size so it is clear on high DPI displays -->
<img src="Logo.png" onload="this.width/=2;this.onload=null;" />
```

NOTE: The reason the logo is scaled to 50% of its original size is that is looks better on high DPI displays, e.g., Windows in high DPI mode, Apple retina displays (Macs, iPhones, iPads etc.).

Stylesheet

All styles used on the application form as stored in Styles.css.

Note the following:

- Date input dates are assigned an 'is-input-date' class.
  - This class is also used in the page's JavaScript code to attach the popup calendar (discussed later).
- The widths of each input field is defined in the stylesheet rather than as an inline style, e.g.:
  - `#txtPostcode {width:6em;}`

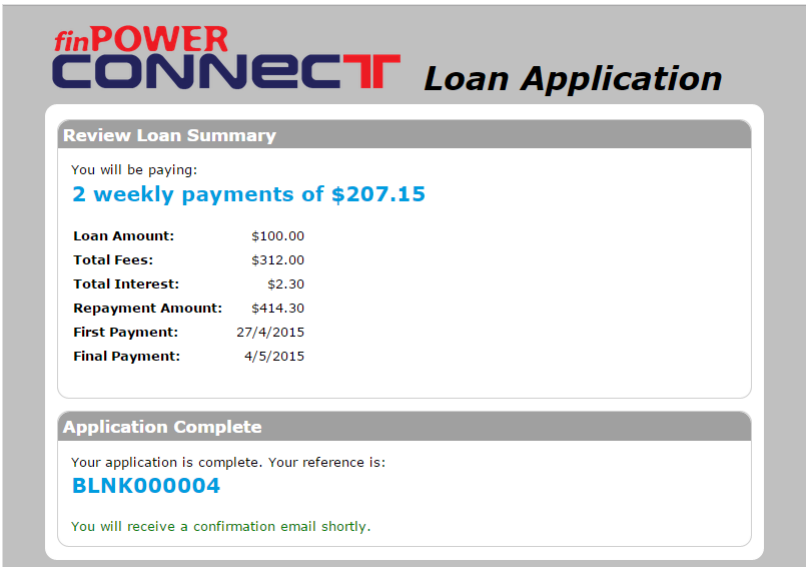
This stylesheet also has a sample section to make the application page 'Responsive'.

NOTE: Response Web pages change their styles (often including their layout) based on the device or screen size that the page is being viewed at.

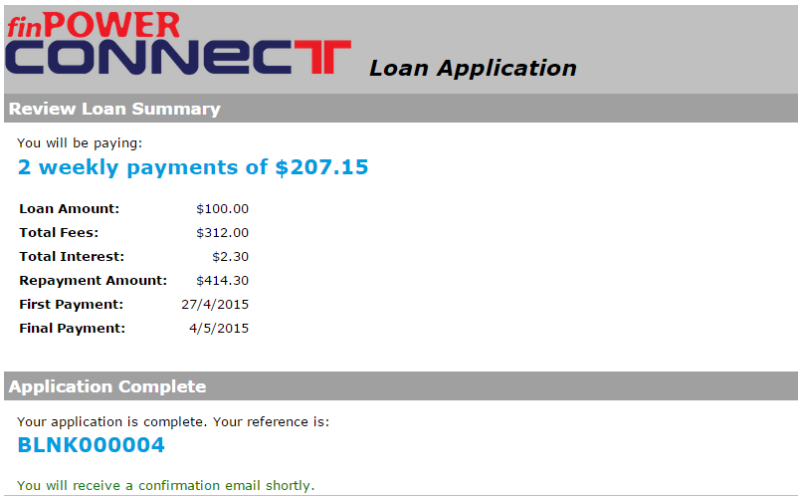
The page styles are altered based on a media rule that invokes the enclosed styles when the screen size falls to 800 pixels or below:

```
/* Responsive */
@media screen and (max-width:800px)
{
  body {margin:0; padding:0.4em 0 0 0;}

  .is-header {margin:0; padding:0;}
  .is-header > h1 {font-size:1.4em;}

  .is-body {margin:0; padding:0; border-radius:0;}

  .is-page {border-radius:0; border:none; border-right:none;}
  .is-page > h1 {border-radius:0;}
  .is-page > .is-form {padding-bottom:0.2em;}

  .is-page .is-form-column {display:block; margin-right:0;}
}
```

This causes the layout to change from:



To the following when the browser window is made smaller or the page is viewed on a small device:



NOTE: This is just a small example of how responsive styles can be used to affect a Web page.

Generally, the idea is to re-style the page to make better use or available screen-estate on smaller device and also to lay it out in a more touch-friendly manor.

HTML and JavaScript

This sample is a single Web page with all HTML and JavaScript code defined in the LoanApplication1VB.aspx file.

The following external libraries are also used:

- jQuery
- jQuery UI
  - Only used for the 'datepicker' popup calendar used for date entry.
- Google Maps
  - Used for address lookups.

NOTE: All external libraries are linked directly from Google rather than being supplied with the sample.

HTML

The HTML follows a simple layout. CSS classes are applied to elements where applicable and inline styles are kept to a minimum (generally just to hide elements by default).

Since the page uses JavaScript calls to perform AJAX calls to ASP.NET Web Methods defined behind the page (discussed in a later section), the following block is included so that Microsoft's Script Manager can be used (this makes it a little easier to make AJAX calls):

```
<!-- BEGIN Script Manager -->
<form id="form1" runat="server" style="display:none">
  <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePageMethods="true"></asp:ScriptManager>
</form>
<!-- END Script Manager -->
```

The page heading (including the logo) is next:

```
<div class="is-header">
  <!-- Scale logo to 50% native size so it is clear on high DPI displays -->
```

```
  <img src="Logo.png" onload="this.width/=2;this.onload=null;" />
  <h1>Loan Application</h1>
</div>
```

As discussed in the Logo section, the logo image is scaled to 50% of its actual size via JavaScript defined in an inline `onload` event.

The main application form is enclosed in a div element with a CSS class of 'is-body'.

Each application page is defined by a div element with a class of 'is-page' and all input fields within a div with a class of 'is-form', e.g.:

```
<!-- BEGIN Verify page -->
<div id="pageVerify" class="is-page" style="display:none">

  <h1 id="lblVerifyHeading">Verify Your ?</h1>

  <div class="is-form">

    <p id="lblVerifyDetails">Please enter your...</p>

    <input id="txtVerificationContact" class="is-input-text is-mandatory" maxlength="100" />

    <label></label>
    <button id="cmdVerify">Verify</button>
    <span id="lblVerificationSent" style="display:none">A verification code has been sent.</span>
    <div id="lblWarningVerify" class="is-warning" style="display:none"></div>
  </div>

</div>
<!-- END Verify page -->
```

NOTE: All pages except the first page (pageLoanDetails) are hidden using the following inline CSS style:

```
style="display:none"
```

The pages are then shown when required.

HTML label elements are used to define the caption for input fields and these define a 'for' property that relates them to the input control. This means that when the label is clicked, the correct input control will take focus, e.g.:

```
<label for="dateClientNextPayDate">Next pay date</label>
<input id="dateClientNextPayDate" type="text" class="is-input-date" />
```

Any input field that is required has a CSS class of 'is-mandatory', as does it's label, e.g.:

```
<label for="numPreferredLoanAmount" class="is-mandatory">How much do you want to borrow?</label>
<input id="numPreferredLoanAmount" class="is-input-number is-mandatory" />
```

JavaScript code (discussed in the next section) adds a red asterisk after all required labels, e.g.:

How much do you want to borrow? *

100.00

JavaScript Overview

The sample makes heavy use of JavaScript including the external jQuery library.

NOTE: For the sake of creating a self-contained sample, all JavaScript is defined within the head section of the HTML page.

If is common to separate out JavaScript (particularly any common, helper methods) into separate .js files.

The JavaScript first defines some constants to make it easy to tweak the sample, e.g.:

```
// Constants
var DATE_FORMAT = "d/m/yy";
var VERIFICATION_METHOD = "Email";  // Email, SMS or blank

var MIN_WEEKLY_PAY = 400;           // Minimum allowable weekly pay
var MAX_LOAN_PERCENT = 1.5;         // Maximum loan amount as a percentage of weekly pay
```

The initialisation code is defined within a jQuery block so that it is only run when the page is fully loaded:

```
// ==========
// Initialise
// ==========
$(function () {
  // All initialisation code goes here
});
```

The initialisation code does the following:

- Initialises date fields to use the jQuery UI date picker.
- Adds a red asterisk after all required fields.
- Configures the 'Verify' page.
  - Text will vary based upon whether an email address of mobile phone number is being used for verification.
- Handles button clicks.
- Initialises address auto-completion (using Google Maps).
- Sets defaults, e.g.:
  - Default loan amount.
- Handles other events, e.g.:
  - Updates the 'After tax pay' label whenever the payment frequency changes.
- Focuses on the first input field.

JavaScript AJAX Calls

The following JavaScript function make AJAX calls to the Web Methods:

- `GetCalculation`
- `SendVerificationCode`
- `Apply`

NOTE: These functions are named after the Web Service that they call. This is just a convention used in this sample.

Each of these functions follows exactly the same structure:

1. Hide any warning message elements, e.g.:

2. Create the JSON object to send to the Web Method.
3. Validate:
    a. Mandatory fields
    b. Other validation, e.g., maximum loan amount.
4. If validation failed then update and show the warning message element and focus on the field that failed validation.
5. Disable the button that was clicked.
6. Call the Web Method:
    a. If this is successful then:
        i. Enable the button that was clicked.
        ii. Do other stuff such as showing the next page or displaying a loan summary.
    b. If this failed then:
        i. Show the error returned using the warning message element.
        ii. Enable the button that was clicked.

NOTE: Calling of the Web Methods is performed via the `PageMethods` object generated by ASP.NET in the Script Manager HTML block.

This is not strictly necessary (and does add a little overhead to the page) but it does make AJAX calls simpler to implement.

JavaScript Helper Functions

Several helper methods exist to get and format values, e.g., get the value of a field as a currency value or format a value as a currency value (including adding the dollar symbol).

The `GetDateOnly` function allows a JavaScript Date value to be returned with a UTC time of midnight. This is essential when passing dates to Web Services (e.g., the applicant's next pay date) since without this, the date will be formatted including a time zone offset which may affect the date received by the Web server.

ASP.NET Web Methods

AJAX calls are made from the client-side JavaScript to Web Methods defined in the LoanApplication1.VB.aspx.vb file.

The page defines the following constants to configure Web Service access and also the Custom Web Service Script:

```
' Debug (better error messages returned)
Const DEBUG As Boolean = True

' Connection Constants
Public Shared WEB_SERVICES_URL As String = "*"
Const WEB_SUBSCRIBER_ID As String = "DEMO"
Const WEB_SUBSCRIBER_SECRET_KEY As String = "11111111111111111111111111111111"
Const USER_ID As String = "admin"
Const USER_PASSWORD As String = "admin"

' Other Constants
Const SCRIPT_ID As String = "LAP1"
```

NOTE: The `WEB_SERVICES_URL` is defined as "*". This simply points the sample to the Web Services instance in which it is running (the shared constructor resolves this).

You will need to change this to a proper URL when creating your own project, e.g.:

http://localhost:51149/ws2/api/

All the Web Methods do it to authenticate against Web Services (to get a Session Token) and then pass on the JSON object to the custom Web Service. Hence, the only code in each of the Web Methods is the following:

```
<System.Web.Services.WebMethod()>
Public Shared Function GetCalculation(calculationDetailsJson As String) As Object

  Return CallCustomWebService("GetCalculation", calculationDetailsJson)

End Function
```

Which simply calls the `CallCustomWebService` method:

```
Private Shared Function CallCustomWebService(action As String,
                                             json As String) As Object

  Dim ErrorCode As String
  Dim ErrorMessage As String
  Dim ErrorMessageInternal As String
  Dim Ok As Boolean
  Dim SessionToken As String
  Dim RequestUrl As String
  Dim ResponseText As String
  Dim StatusCode As Integer

  ' Assume Success
  Ok = True

  ' Connect to Web Services
  Ok = ExecuteAuthenticateUser(SessionToken, ErrorMessage)

  ' Call Custom Web Service to Get Calculation
  If Ok Then
    ' Build Request URL
    RequestUrl = WEB_SERVICES_URL & String.Format("Custom/{0}?action={1}", SCRIPT_ID, action)

    ' Execute Request
    If ExecuteRequest(RequestUrl, "POST", "JSON", json, Nothing, SessionToken, ResponseText,
                      Nothing, StatusCode, ErrorMessage, Nothing) AndAlso
                      StatusCode = System.Net.HttpStatusCode.OK Then
      ' OK
    ElseIf Len(ErrorMessage) <> 0 Then
      ' Failed Executing Request
      Ok = False
    Else
      ' Parse Error Response (JSON format)
      Ok = False
      ParseErrorJson(ResponseText, ErrorCode, ErrorMessage, ErrorMessageInternal)

      ' Pass on the Status Code and modify the Error Message (based on the ErrorCode) if required
      Select Case ErrorCode
        Case Else
          ' Pass on the unmodified Error Message returned from the Custom Web Service
      End Select
    End If
  End If

  If Ok Then
    ' Return ResponseText (this is the JSON returned from the Custom Web Service)
    Return ResponseText
  Else
    ' Error
    Throw New ApplicationException(ErrorMessage)
  End If

End Function
```

WARNING: When running in debug mode in a development environment, e.g., from Visual Studio, the above code will stop when it hits the `Throw New ApplicationException` line. Simply press F5 to continue.

NOTE: When running in a production environment, you may need to add the following line to your web.config file so that when an Exception is thrown, the correct Error Message is returned to the Client-side JavaScript (by default, the generic 'There was an error processing the request.' message may be returned):

```
<system.web>
  <customErrors mode="Off"/>
```

Custom Web Services

Importing the LoanApplication1VB.xml file into the finPOWER Connect Scripts library will create a new 'Custom Web (Web API)' type Script with a code of 'LAP1'.

This Script is explained fully in this section.

Constants

The sample contains many constants.

The following constants must always be defined:

- `CreateAccountApp`
  - Indicates whether to create an Account Application instead of a 'Quote' Account.
- `AccountTypeId`
  - The Id of the Account Type to use when creating 'Quote' Accounts and also when adding a 'Quote' to an Account Application.

The following constants must be defined if creating an Account Application:

- `AccountAppTypeId`
  - The Id of the Account Application Type to create.

The following constants must be defined if creating a 'Quote' Account:

- `AccountRoleId`
  - The Id of the Account Role to assign to the borrower.
- `ClientGroupId`
  - The Id of the Client Group to use when creating the Client record for the Account.
- `ClientTypeId`
  - The Id of the Client Type to use when creating the Client record for the Account.
- `ContactMethodAddress`, `ContactMethodEmail`, `ContactMethodMobile`, `ContactMethodPhone`
  - The Id of the Contact Methods to use when creating Contact Methods for the Client record.

The follow constants can optionally be updated to determine the verification method and confirmation method to use during the online application:

- `VERIFICATION_METHOD`

- As part of the online application, you can force the applicant to enter a verification code that is sent to them. This constant defines which verification method to use:
  - Email
    - An Email will be sent to the applicant.
    - **NOTE:** This requires that an SMTP server is defined under either Global Settings or the Web Configuration being used by the Web Services.
  - SMS
    - A text message will be sent to the applicant.
    - **NOTE:** This requires that the finPOWER Connect database is licensed for the SMS Add-On and that SMS details are configured.
    - **WARNING:** Sending SMS messages has an associated cost.
  - blank
    - No verification will be sent to the applicant.
- **NOTE:** The LoanApplication1.aspx also defines a `VERIFICATION_METHOD` constant that must match this.

- `CONFIRMATION_METHOD`
  - Upon finishing the online application, you can send a confirmation message to the applicant. This constant defines how to send the message:
    - Email
    - SMS
    - Blank
      - No confirmation message will be sent.

Configuration Validation

The first block of code in the Main method validates constants and licensing options.

NOTE: It is always good practice to validate as much as possible before even attempting to do anything else.

This makes it easier to avoid issues in the Script code such as attempting to access an Account Type that does not exist.

Multiple Web Services in a single Script

Partially to keep this sample self-contained and partially because it just makes sense, the 'LAP1' Custom Web Services Script actually handles multiple Web Services:

- GetCalculation
  - Perform a loan calculation and return the results.
- SendVerificationCode
  - Send the applicant a 'Verification Code' to either their email or mobile phone.
- Apply
  - Add an application (either as a 'Quote' Account or an Account Application).

To enable the Script to handle multiple Web Services, the Script accepts an 'Action' parameter and simply has a Select Case to handle the various actions, i.e.:

```
' Get Parameters
If Ok Then
  Action = request.Parameters.GetString("Action")
End If
```

```
' Handle Action (this Web Service can handle several different actions)
If Ok Then
  ' Handle Action
  Select Case Ucase(Action)
    Case "GETCALCULATION"
      ' Get Calculation
      Return Action_GetCalculation(request)

    Case "SENDVERIFICATIONCODE"
      ' Send Verification Code
      Return Action_SendVerificationCode(request)

    Case "APPLY"
      ' Add Application
      If CreateAccountApp Then
        Return Action_ApplyAccountApp(request)
      Else
        Return Action_ApplyAccount(request)
      End If

    Case Else
      ' Unknown
      Return request.CreateErrorResponse(HttpStatusCode.BadRequest, String.Format("Failed to
execute custom Web Service Script '{0}'. Action '{1}' is not handled.", ScriptInfo.ScriptId,
Action), "Action.Unhandled", "")
  End Select
Else
```

Deserialising JSON Requests

All services in the Script assume that the request is a JSON-formatted string.

Proxy objects for each request are defined at the end of the Script, e.g.:

```
' ====================================================
' Classes used for JSON serialisation/ deserialisation
' ====================================================
Public Class clsCalculationDetails

  ' Calculation Details
  Public ClientPayFreq As String
  Public ClientNextPayDate As Date?
  Public ClientPayAmount As Decimal
  Public PreferredLoanAmount As Decimal
  Public PreferredRepayments As Integer

End Class
```

The finPOWER Connect business layer contains functionality for deserialising the request into an object, e.g.:

```
If finBL.Runtime.WebUtilities.DeserialiseJsonStringToObject(request.RequestText,
                                                            GetType(clsCalculationDetails), Obj) Then
  CalculationDetails = DirectCast(Obj, clsCalculationDetails)
Else
  Ok = False
End If
```

NOTE: Many proxy object properties use nullable types, e.g., `ClientNextPayDate As Date?`

The Script must handle nullable properties correctly, i.e., check the property has a value and cast it to the correct date type, e.g.:

```
If calculationDetails.ClientNextPayDate IsNot Nothing Then
  ' Set first payment date to next pay date
  If CDate(calculationDetails.ClientNextPayDate) >= .OpeningDate Then
```

Serialising Response as JSON and Avoiding Date Issues

All services in the Script return objects. The Microsoft Web API will automatically serialise these as JSON text.

Objects for each response are defined at the end of the Script, e.g.:

```
Public Class clsCalculationResult

  ' Loan Calculation Result
  Public LoanAmount As Decimal
  Public TotalFees As Decimal
  Public TotalInterest As Decimal
  Public RepaymentAmount As Decimal
  Public PaymentCount As Integer
  Public PaymentRegular As Decimal
  Public FirstPaymentDate As Date
  Public FinalPayment As Decimal
  Public FinalPaymentDate As Date
  Public PaymentFrequency As String

End Class
```

And are returned from the Script as follows:

```
Return request.CreateResponse(HttpStatusCode.OK, CalculationResult)
```

If dates are returned directly from the business layer, e.g.:

```
account.Calculation.GetPaymentRegularFirst(FirstPaymentDate, PaymentRegular, False)
CalculationResult.FirstPaymentDate = FirstPaymentDate
```

The date MAY (depending on how it is held in the finPOWER Connect business layer) be returned including a time portion or offset as the following JSON response (sent from a Web Server in New Zealand) demonstrates:

```
{"LoanAmount":1000.0,
"TotalFees":312.00,
"TotalInterest":12.51,
"RepaymentAmount":1324.51,
"PaymentCount":4,
"PaymentRegular":331.13,
"FirstPaymentDate":"2015-04-27T00:00:00+12:00",
"FinalPayment":331.12,
"FinalPaymentDate":"2015-05-18T00:00:00+12:00",
"PaymentFrequency":"Weekly"}
```

This will cause confusion at some point, e.g., if the applicant is in a different time zone to the Web Server.

What we should do (as detailed in the Dates section of the Web Services API Reference) is to convert the dates accordingly, i.e.:

```
CalculationResult.FirstPaymentDate = finBL.Runtime.DateUtilities.CastToUtcDate(FirstPaymentDate)
```

This will cause the date to be returned in UTC format with no time offset, e.g.:

```
{"LoanAmount":1000.0,
"TotalFees":312.00,
"TotalInterest":12.51,
"RepaymentAmount":1324.51,
"PaymentCount":4,
"PaymentRegular":331.13,
"FirstPaymentDate":"2015-04-27T00:00:00Z",
"FinalPayment":331.12,
"FinalPaymentDate":"2015-05-18T00:00:00Z",
"PaymentFrequency":"Weekly"}
```

WARNING: If you attempt to use JavaScript to deserialise and display these dates in an HTML page, you must also be aware of possible time zone issues. This is discussed later in this section.

Action_GetCalculation method

This method performs an Account calculation and returns the result.

This is achieved as follows:

- Deserialise the request into a `clsCalculationDetails` object.
- Create a `finAccount` object and update this from the request:
  - This calls the `UpdateAccountFromCalculationDetails` method (next section) which also validates the request.
- Returns a `clsCalculationResult` response.

UpdateAccountFromCalculationDetails method

This method validates the passed in Calculation Details and updates the supplied Account object.

This is achieved as follows:

- Validate some basic calculation properties.
- Update the `finAccount.Calculation` object.
  - This includes setting the First Payment Date if the applicant's next pay date was supplied and is after the Account's Opening Date.
- Calculate the loan using the `finAccount.Calculation.Calculate()` method.

NOTE: In this sample, the maximum loan amount and the applicant's pay after tax are not validated in the Script; they are only validated in the client-side JavaScript code before calling the Web Service.

This is mainly to keep the sample simpler. In a real-life Web Service, any validation performed on the client-side (e.g., from JavaScript), should also be performed on the server-side (i.e., in the Web Service Script).

Action_SendVerificationCode method

This method sends a verification code to the applicant.

The verification code is sent to either an email address or mobile phone number depending on the `VERIFICATION_METHOD` constant.

This is achieved as follows:

- First, check that verification is being used, i.e., `VERIFICATION_METHOD` is not blank.
- Deserialise the request into a `clsSendVerificationDetails` object.
  - The `Code` property or this object is simply a non-static string. In this case, the date and time on the applicant's PC formatted as a string.
  - The `VerificationContact` property is the applicant's email address of mobile phone number.
- Calls the `CreateVerificationCode` method (next section) to generate a 6 character verification code.
- Sends the verification code to the applicant's email or mobile number.
  - The email message is formatted as plain text for ease of copying and pasting.

- Returns a simple OK response.

CreateVerificationCode method

This method generates a simple verification code based on `code` and `verificationContact` parameters.

> **NOTE:** The verification code is not designed to be a secure code, just something reasonably unique that the end-user can easily type into a Web page.

In the sample, the `code` parameter is set by client-side JavaScript code and is simply the date and time formatted as a string, e.g..:

```
{"Code":"\"2015-04-20T02:04:09.279Z\"","VerificationContact":"test@intersoft.co.nz"}
```

> **NOTE:** The code could be anything but, by using the date and time, a different code will be generated for the same email address of mobile phone number.

The `verificationContact` is either the applicant's email address or mobile phone number.

The following code generates a 6 letter verification code. The letters included are defined in a `CodeChars` variable and consist of letters that are easy to read.

```
Private Function CreateVerificationCode(code As String,
                                        verificationContact As String) As String

  Dim CodeChars As String = "AEHJMNOPRSTUWXYZ"

  Dim VerificationCode As String
  Dim i As Integer
  Dim j As Integer
  Dim strTemp As String

  ' Create unsalted Hash
  strTemp = finBL.Runtime.Encryption.CreateUnsaltedHash(code & verificationContact & "123456",
                                                        "vercode")

  ' Now create a nice, user-friendly 6 character code
  For i = 1 To 6
    ' Rotate the code string
    For j = 0 To Asc(Mid(strTemp, i, 1))
      CodeChars = Mid(CodeChars, 2) & Left(CodeChars, 1)
    Next

    ' Add to verification code
    VerificationCode &= Left(CodeChars, 1)
  Next

  Return VerificationCode

End Function
```

The code does the following:

- Generates a hash of the code and verificationContact parameters.
  - 6 characters are added to the end, just in case.
  - A key of "vercode" is used. This could be anything.
- The code then loops 6 times:
  - Rotates the CodeChars variable based on the ASCII code of the nth letter of the hash.
  - Appends this character to the verification code.

NOTE: This code is just a simple example of how you might generate a verification code.

Action_ApplyAccount method

This method creates a 'Quote' Account and optionally sends a confirmation email message or SMS.

Points to note are:

- The `Action_ApplyValidateRequest` method is used to validate the request.
  - This is a separate function since it is used whether creating an Account or an Account Application.
- Creates a Client and Account record within a database transaction.
  - If either fail to save, the database transaction is rolled back.
  - Uses the `UpdateAccountFromCalculationDetails` method to update the Account Calculation.
    - This is the same function used in the by the `Action_GetCalculation` method and is also used when creating an Account Application.
- Optionally sends a confirmation email or SMS message.
  - Based on the `CONFIRMATION_METHOD` constant.
  - The Web Service will not fail if no confirmation is send.
    - Instead, `ConfirmationEmailSent` and `ConfirmationSmsSent` properties are updated on the response object (`clsAddApplicationResult`).
      - The client-side JavaScript can then use these to display varying messages.
  - The `CreateConfirmationMessage` and `CreateConfirmationSubject` methods are used to generate the confirmation message.
- The `Reference` property on the response is set to the new Account's Id.

Action_ApplyValidateRequest method

This method validates the request send to the Apply Web Service.

The code does the following:

- Validates mandatory properties such as `FirstName` and `LastName`.
- Validates that the date of birth resolves to an age between 18 and 99.
  - This also helps prevent data-entry errors such as the applicant entering a year of 1876.
- Validates the verification code:
  - This uses the `CreateVerificationCode` method but:
    - Instead of the email address or mobile number passed to the SendVerificationCode Web Service, the email address or mobile number send to the Apply Web Service is used.
      - This prevents the applicant from supplying a different email address or mobile number for verification versus the one they enter on the actual application.

CreateConfirmationMessage method

This method generates an email or SMS confirmation message.

The `reference` parameter is either the Account Id or Account Application Id.

The `branchPk` parameter is required since we are including the built-in email signature which is Entity-specific.

If an email confirmation is being generated then note the following:

- The message is formatted as HTML.
  - The `Apply_SendVerificationCode` method generates a plain text email message since this should not typically include much information and should be easy for the applicant to copy and paste into the HTML application form.
  - The message is generated using the `finBL.DocumentFunctions.FormatEmailMessage` method. This does the following:
    - Adds surrounding HTML tags and styles (since the `htmlMessageTemplate` parameter is set to "*").
    - Adds the current User's signature (since the `signature` parameter is set to "*").
      - **NOTE:** This is for example's sake but may not make sense since the "current User" is the `USER_ID` hard-coded as a constant at the top of the LoanApplication1.aspx.vb file.

If you would like to generate an email message not using any of the built-in style or signature information, this can easily be achieved as per the following example:

```
' Email (HTML)
strTemp = ""
strTemp &= "<div style='font:14pt Verdana'>"
strTemp &= "<p>Thank you for your application.</p>"
strTemp &= String.Format("<p>Your reference is <b>{0}</b>.</p>", finBL.HtmlEncode(reference))
strTemp &= "</div>"
Return strTemp
```

NOTE: When generating HTML emails, it is advisable to use inline styles and to test the email on various platforms.

Some products such as Microsoft Outlook have a habit of playing around with HTML formatted emails so they may not appear as expected.

CreateConfirmationSubject method

As per the `CreateConfirmationMessage` method but simply returns a subject for email messages.

Troubleshooting

Timeout when Authenticating Client

A timeout error when attempting to authenticate a Client via the Authentication/AuthenticateClient service may be due to the following:

Misconfigured Address Database

If the Address database being used by the finPOWER Connect business layer is not available, attempting to connect to it may cause a time out.

This may be an issue when attempting to authenticate as a Client since the response from this service includes formatted Branch address details which involves accessing the Addressing interface which will always attempt to initialise a connection to the Address database when first accessed.