Using Apple's Global Service Exchange web service in FileMaker

Quite a few Apple shops use FileMaker and/or Xojo for their development of in-house tools. A common request is to use Apple's webservices to query warranty status. So today I want to show some scripts on how to do this. First of course you have to ask Apple for a GSX login which may require some paperwork. Next you need to white list your static IP for their webservice and get the credentials.

You request a certificate from Apple, so you generate a private key. The tricky key is to copy the private key with the certificate into one pem file. This pem file is than used wiht our script. Also please download a standard cacert.pem file with root certificates.
The first script logs into the webservice from Apple. We pass the user id and service account id in the login request.

#Parameters
Set Variable [$path; Value:"/path to files/"]
Set Variable [$userid; Value:"xxx"]
Set Variable [$serviceAccountNo; Value:"yyy"]
Set Variable [$PrivateKeyPassword; Value:"secret password"]
#Build the XML
Set Variable [$xml; Value:GSX::Login XML Template]
Set Variable [$xml; Value:Substitute($xml; "$userId$"; MBS("Text.EncodeToXML"; $userid))]
Set Variable [$xml; Value:Substitute($xml; "$serviceAccountNo$"; MBS("Text.EncodeToXML"; $serviceAccountNo))]
#Start curl session
Set Variable [$curl; Value:MBS("CURL.New")]
Set Variable [$r; Value:MBS("CURL.SetOptionURL";$curl; "https://gsxapi.apple.com:443/gsx-ws/services/emea/asp")]
#We use a PEM file with private key and our certificate
Set Variable [$r; Value:MBS("CURL.SetOptionSSLCertType"; $curl; "PEM")]
Set Variable [$r; Value:MBS("CURL.SetOptionKeyPassword"; $curl; $PrivateKeyPassword)]
Set Variable [$r; Value:MBS("CURL.SetOptionSSLCert"; $curl; $path & "your.pem")]
#and the usual root certificates
Set Variable [$r; Value:MBS("CURL.SetOptionCAINFO"; $curl; $path & "cacert.pem")]
#Use TLS v.1.2
Set Variable [$r; Value:MBS("CURL.SetOptionSSLVersion"; $curl; 6) // TLS v1.2]
#Wait for 10 seconds
Set Variable [$r; Value:MBS("CURL.SetOptionTimeOut"; $curl; 10)]
#Specify content type and SOAP Action for webservice

```
Set Variable [$r; Value:MBS("CURL.SetOptionHTTPHeader"; $curl; "Content-Type: text/xml; charset=UTF-8"; "SOAPAction: \"urn:authenticate\"")]
#Pass XML to CURL
Set Variable [$r; Value:MBS("CURL.SetOptionPostFields"; $curl; $xml)]
#Run the request
Set Variable [$r; Value:MBS("CURL.Perform"; $curl)]
#Check result by storing result, debug log and output data in fields
Set Variable [$httpResponse; Value:MBS( "CURL.GetResponseCode"; $curl )]
Set Field [GSX::CURL Result; $r]
Set Field [GSX::CURL Input; $xml]
Set Field [GSX::CURL Debug; MBS("CURL.GetDebugAsText"; $curl; "UTF8")]
Set Field [GSX::CURL Output; MBS("CURL.GetResultAsText"; $curl; "UTF8")]
If [$r = "OK" and $httpResponse = 200]
        #on success, get our session id
        Set Field [GSX::SessionID; MBS("Text.FindBetween"; GSX::CURL Output; "<userSessionId>"; "</userSessionId>")]
End If
#always clean up
Set Variable [$r; Value:MBS("CURL.Cleanup")]
```

Here is the login template:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:glob="http://gsxws.apple.com/elements/global">
  <soapenv:Header/>
  <soapenv:Body>
    <glob:Authenticate>
      <AuthenticateRequest>
        <userId>$userId$</userId>
        <languageCode>de</languageCode>
        <userTimeZone>CET</userTimeZone>
        <serviceAccountNo>$serviceAccountNo$</serviceAccountNo>
      </AuthenticateRequest>
    </glob:Authenticate>
  </soapenv:Body>
</soapenv:Envelope>
```

Next we run a second query to check the warranty status of an iPhone. Here we pass in the session ID we got above, the serial number of the iPhone and your shop number. The result is a xml where we extract again the interesting values like the status and configuration information:

```
#Parameters
Set Variable [$path; Value:"/path to files/"]
Set Variable [$PrivateKeyPassword; Value:"secret password"]
#Build the XML
Set Variable [$xml; Value:GSX::Query XML Template]
Set Variable [$xml; Value:Substitute($xml; "$sessionid$"; MBS("Text.EncodeToXML"; GSX::SessionID))]
```

```
Set Variable [$xml; Value:Substitute($xml; "$serialnumber$"; MBS("Text.EncodeToXML"; GSX::SerialNo:))]
Set Variable [$xml; Value:Substitute($xml; "$shipto$"; MBS("Text.EncodeToXML"; "xxxx"))]
#Start curl session
Set Variable [$curl; Value:MBS("CURL.New")]
Set Variable [$r; Value:MBS("CURL.SetOptionURL";$curl; "https://gsxapi.apple.com:443/gsx-ws/services/emea/asp")]
#We use a PEM file with private key and our certificate
Set Variable [$r; Value:MBS("CURL.SetOptionSSLCertType"; $curl; "PEM")]
Set Variable [$r; Value:MBS("CURL.SetOptionKeyPassword"; $curl; $PrivateKeyPassword)]
Set Variable [$r; Value:MBS("CURL.SetOptionSSLCert"; $curl; $path & "your.pem")]
#and the usual root certificates
Set Variable [$r; Value:MBS("CURL.SetOptionCAINFO"; $curl; $path & "cacert.pem")]
#Use TLS v.1.2
Set Variable [$r; Value:MBS("CURL.SetOptionSSLVersion"; $curl; 6) // TLS v1.2]
#Wait for 10 seconds
Set Variable [$r; Value:MBS("CURL.SetOptionTimeOut"; $curl; 10)]
#Specify content type and SOAP Action for webservice
Set Variable [$r; Value:MBS("CURL.SetOptionHTTPHeader"; $curl; "Content-Type: text/xml; charset=UTF-8"; "SOAPAction: \"urn:warrantyStatus\"")]
#Pass XML to CURL
Set Variable [$r; Value:MBS("CURL.SetOptionPostFields"; $curl; $xml)]
#Run the request
Set Variable [$r; Value:MBS("CURL.Perform"; $curl)]
#Check result by storing result, debug log and output data in fields
Set Variable [$httpResponse; Value:MBS( "CURL.GetResponseCode"; $curl )]
Set Field [GSX::CURL Result; $r]
Set Field [GSX::CURL Input; $xml]
Set Field [GSX::CURL Debug; MBS("CURL.GetDebugAsText"; $curl; "UTF8")]
Set Field [GSX::CURL Output; MBS("CURL.GetResultAsText"; $curl; "UTF8")]
If [$r = "OK" and $httpResponse = 200]
    #on success, get our values
    Set Field [GSX::Status; MBS("Text.FindBetween"; GSX::CURL Output; "<warrantyStatus>"; "</warrantyStatus>")]
    Set Field [GSX::coverageEndDate; MBS("Text.FindBetween"; GSX::CURL Output; "<coverageEndDate>"; "</coverageEndDate>")]
    Set Field [GSX::coverageStartDate; MBS("Text.FindBetween"; GSX::CURL Output; "<coverageStartDate>"; "</coverageStartDate>")]
    Set Field [GSX::daysRemaining; MBS("Text.FindBetween"; GSX::CURL Output; "<daysRemaining>"; "</daysRemaining>")]
    Set Field [GSX::estimatedPurchaseDate; MBS("Text.FindBetween"; GSX::CURL Output; "<estimatedPurchaseDate>"; "</estimatedPurchaseDate>")]
    Set Field [GSX::purchaseCountry; MBS("Text.FindBetween"; GSX::CURL Output; "<purchaseCountry>"; "</purchaseCountry>")]
    Set Field [GSX::registrationDate; MBS("Text.FindBetween"; GSX::CURL Output; "<registrationDate>"; "</registrationDate>")]
    Set Field [GSX::productDescription; MBS("Text.FindBetween"; GSX::CURL Output; "<productDescription>"; "</productDescription>")]
    Set Field [GSX::configDescription; MBS("Text.FindBetween"; GSX::CURL Output; "<configDescription>"; "</configDescription>")]
```

```
        Set Field [GSX::activationLockStatus; MBS("Text.FindBetween"; GSX::CURL
Output; "<activationLockStatus>"; "</activationLockStatus>")]
End If
#always clean up
Set Variable [$r; Value:MBS("CURL.Cleanup")]
```

The call is the same, except that we use a different template and a different SOAP Action.
The query template:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:glob="http://gsxws.apple.com/elements/global">
  <soapenv:Header/>
  <soapenv:Body>
    <glob:WarrantyStatus>
      <WarrantyStatusRequest>
        <userSession>
          <userSessionId>$sessionid$</userSessionId>
        </userSession>
        <unitDetail>
          <serialNumber>$serialnumber$</serialNumber>
          <shipTo>$shipto$</shipTo>
        </unitDetail>
      </WarrantyStatusRequest>
    </glob:WarrantyStatus>
  </soapenv:Body>
</soapenv:Envelope>
```

If you wonder where we got the XML templates from, you can of course simply copy them from Apple's documentation, build them from the wsdl or better use an application like the free SOAPUI tool.

I hope this helps anyone trying to use GSX with FileMaker. Please use the test service before switching to production service. If you have questions, please do not hestiate to contact us.
For Xojo the scripts should easily translate to Xojo code. If you need assistance, you can call me.