1. Upload eines ASCII-Files auf einen HTTP-Server von UPS

Vortrag von Jürgen Asal für die VO UG Bodensee (http://www.voug-bodensee.net/) vom 25.10.2007.

Er basiert auf dem bekannten Vortrag von Norbert Kolb „Get or Post“, welcher unter http://www.gsiberg.net/ak/download/GetOrPost.zip downgeladen werden kann.

1.1. Grundsätzlicher Programmablauf

1.1.1. Aufgabe:

Es soll eine ASCII-Datei an folgende URL hochgeladen werden und die Antwort muss ausgewertet werden:

https://www.pld-certify.ups.com/hapld/tos/kdwhapltos

1.1.2. Besonderheiten:

1. Es muss eine Verbindung mit SSL aufgebaut werden
2. Es ist eine Anmeldung an den Server mittels User-ID und Passwort notwendig

1.1.3. Grundsätzlicher Ablauf eines HTTP-Uploads

1. Upload-Datei erzeugen als String
2. Initialisieren der cHttp - Klasse
3. Verbindung zum Host aufbauen mit ConnectRemote
4. Aufbau einer POST - Verbindung mit OpenRequest
5. Upload-Datei absenden mit SendRequest
6. Antwort einlesen mit GetResponse
7. Auswerten der Antwort

1.2. Implementation

1.2.1. URL aufteilen

Die URL muss aufgeteilt werden:

https://www.pld-certify.ups.com/hapld/tos/kdwhapltos

Der rote Teil gehört zum „ConnectRemote“, der blaue Teil zum „OpenRequest“

1.2.2. Verbindung zum Server herstellen

Ich habe zuerst versucht, die Verbindung mit folgender URL aufzubauen (wie man es vom Browser kennt):

https://www.pld-certify.ups.com

Das ganze hat lediglich Fehlermeldungen eingebracht

Richtig ist: Aufbau der Verbindung über Post 443 und Angabe der URL OHNE das ‚S’ in ‚https’: http://www.pld-certify.ups.com (Anmerkung von Norbert: man kann das „http:://“ komplett weglassen) . Natürlich muss man ggf. im Firewall den Port 443 öffnen!

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

```
// HTTP-Verbindung via Port 443 (SSL) initialisieren
 oHttp := cHttp{ "UPS", 443 }
 IF oHttp:ConnectRemote( “http://www.pld-certify.ups.com” )
   // Verbindungsaufbau erfogreich
 ELSE
   // Fehler
 ENDIF
```

1.2.3. POST-Verbindung aufbauen

Als nächstes folgt der Aufbau der Post-Verbindung:

```
 IF ! oHttp:OpenRequest("POST", "/hapld/tos/kdwhapltos", INTERNET_FLAG_RELOAD +
   INTERNET_FLAG_SECURE + INTERNET_FLAG_IGNORE_CERT_CN_INVALID +
   INTERNET_FLAG_IGNORE_CERT_DATE_INVALID )

   // Verbindungsaufbau erfogreich
 ELSE
   // Fehler
 ENDIF
```

Entscheidend für den Erfolg sind die untenstehenden Flags. Das Handbuch verweist auf die WIN32-Doku, welche aber keinen Eintrag für diese Flags erhält. Im Internet findet man bei Microsoft unter

http://support.microsoft.com/kb/168151

ein Dokument mit dem genialen Titel **„Wie Machen von SSL-Anfragen mit Hilfe des WinInet“** sowie unter

http://support.microsoft.com/kb/157607/de

ein Dokument mit dem noch genialeren Titel **„DOCERR: FtpGetFile & FtpOpenFile Flags dokumentierte falsch“**

Mit etwas Fantasie kann man den Inhalt verstehen. Diese Artikel verdeutlichen auch eindrucksvoll den aktuellen Stand der automatischen Sprachübersetzung.

INTERNET_FLAG_RELOAD

Bedeutung: funktionieren FtpGetFile-Anrufe unter Verwendung von INTERNET_FLAG_RELOAD, um anzugeben, dass FTP-Dateien von dem Draht und nicht von dem Cache bereits geliefert werden sollten immer falsch, wenn sich OR'ed dieses Flag an dem fünften Parameter befindet.

Auf gut deutsch: Zu sendendes Dokument nicht aus dem Cache auslesen

INTERNET_FLAG_SECURE

Bedeutung: Gibt an, dass es sich um eine SSL - Verbindung handelt

INTERNET_FLAG_IGNORE_CERT_CN_INVALID

Bedeutung: Fehler ignorieren, welche durch ein ungültiges Zertifikat gemeldet werden

INTERNET_FLAG_IGNORE_CERT_DATE_INVALID

Bedeutung: Fehler ignorieren, welche von einem abgelaufenen Server-Zertifikat verursacht werden können.

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

1.2.4. Aufbau der POST-Meldung (Request- und Hearder-String)

Der Aufbau der POST - Meldung erfolgt grundsätzlich wie im Vortrag von Norbert.

Die Vorgabe zum Aufbau liefert UPS.

Zuerst erzeugen wir den Request-String.

Wichtig: Die Anzahl der CRLF ist von absolut entscheidender Bedeutung. Ein CRLF zuviel oder zuwenig, und es funktioniert nicht mehr!

Hier ein vollständiger Request-String:

```
--BOUNDARYCRLF
Content-type: application/x-www-form-urlencodedCRLF
Content-length: 130CRLF
CRLF
AppVersion=1.0&AcceptUPSLicenseAgreement=YES&ResponseType=application/x-ups-
pld&VersionNumber=V4R1&UserId=320001&Password=xxxxxxxxCRLF
CRLF
--BOUNDARYCRLF
Content-type: application/x-ups-binaryCRLF
Content-length: 2357CRLF
CRLF
020020                                  2007102500000591476525300   000000001*AA320007    CH
5914765253000001*BA1Z320007D300070004
00001+00000000000000300+0000000000000000KGS07COL10Jürgen Asal
3CMUSD000001*CA06Jürgen Asal                        Asal Informatik GmbH
Softwarehouse                                                   Buchackerweg 14
Hochwald                          4146     CH0617518030      0617518031
*CA18                                  Ivoclar-Vivadent France            La Chapelle du
Puits                                                Boite Postale 118                  St.
Jorioz                        74410    FR                              123321
*EADCS*FA001TN00001292977                        *GA00603support@asal-info.ch
Asal Informatik GmbH
*IA32000733HGC1Teile aus Gold                                                        *KA
CHUSD+000000000000120000+000000000000120000000001PC
Teile aus Gold
+000000   +000000000000000000
*LA20071025+000000000000120000USD00001292977
+000000000000120000+000000000000120000FOB
+000000000000000000+000000000000000000+000000000000000000+000000000000000000
*PA1Z320007D300070004                 03+0000030
Teile aus Gold                      +0000060+00003000+00003000+00004500*SA000011CRLF
CRLF
--BOUNDARY--CRLF
```

Dunkelgrün ist der zu übermittelnde ASCII-File

Nun brauchen wir noch einen Header-String:

```
Content-Type: multipart/mixed; boundary=BOUNDARYCRLF
Content-Length: 2669CRLF
CRLF
```

Grün ist die Länge des Request-Stringes

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

1.2.5. Absenden des POST-Requests

Diese beiden Strings können nun abgeschickt werden. Dazu brauchen wir zuerst aber einen Buffer:

```
pBuffer := StringAlloc( cRequest )
lError := !oHttp:SendRequest(cHeaders, pBuffer, SLen(cRequest))
```

Keinesfalls vergessen darf man, den Buffer anschliessend zu löschen:

```
MemFree( pBuffer )
```

Wenn kein Error aufgetaucht ist, wird die Antwort des Servers eingelesen:

```
cHtmlResult := oHttp:GetResponse()
```

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

1.2.6. Serverantwort einlesen

Die Serverantwort ist ein String mit HTML-Befehlen, welche man selbst auseinanderpfriemeln darf:

```
--BOUNDARY
Content-type: text/html
Content-length: 138

<HTML>
<HEAD>
<TITLE>UPS Internet Software</TITLE>
</HEAD>
<BODY>
<P>UPS Internet Software, Copyright UPS 1998</P>
</BODY>
</HTML>

--BOUNDARY
Content-type: application/x-ups-psmpld
Content-length: 104

UPSOnLine%1.0%0000%0000Successful completion - No errors found.

--BOUNDARY
Content-type: application/x-ups-pld
Content-length: 800

00020000787300000640 KDGPLCR1                    United Parcel Service
10/25/07  Page:1                    Test PLD Control Report                     05:35 AM
------------------------------------------------------------------------------- Pickup
Date:20071025 Sequence#:000005914765252 #of Segments:000000000000000012
Shipper#:320007     Book/Page#:5914765252 Shipments:000001 Packages:0000000001
10000012900000000Successful completion - No errors found.
!!! Test Upload !!!
File Uploaded: n/a

--BOUNDARY--
```

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

1.3. Vollständiger Sourcecode

```
METHOD UPSXMLRequestHttp( cRequest AS STRING, cXmlResultsFile AS STRING ) AS LOGIC PASCAL
CLASS API_UPS
  //p Führt eine XML-Anfrage an UPS durch
  //a cRequest \tab Anfrage als XML-Datei
  //a cXmlResultsFile \tab Pfad und Name. Die Serverantwort wird in diesem File abgespeichert
  //r lError \tab True, wenn ein Fehler aufgetreten ist.
  //d Der cRequest wird automatisch mit den Anmeldeinformationen ergänzt

  LOCAL oHttp         AS cHttp
  LOCAL pBuffer       AS PTR
  LOCAL cHeaders      AS STRING
  LOCAL cXmlResult    AS STRING       // Antwort der XML-Abfrage
  LOCAL pHandle       AS PTR
  LOCAL lError        AS LOGIC

  ~"Onlyearly-"

  oHttp := cHttp{ "UPS", 443 }

  IF oHttp:ConnectRemote( cHost )

    IF !oHttp:OpenRequest("POST",
cURI,INTERNET_FLAG_RELOAD+INTERNET_FLAG_SECURE+INTERNET_FLAG_IGNORE_CERT_CN_INVALID+INTERNET_F
LAG_IGNORE_CERT_DATE_INVALID )
      cError := "OpenRequest 'POST' to URI: "+cURI+" failed"
      lError := TRUE
    ELSE
      cRequest := ;
        '<?xml version="1.0"?>' + CRLF +;
        '<AccessRequest xml:lang="en-US">' + CRLF +;
        '   <AccessLicenseNumber>' + cXML_Key + '</AccessLicenseNumber>' + CRLF +;
        '   <UserId>' + cUserID + '</UserId>' + CRLF +;
        '   <Password>' + cPassword + '</Password>' + CRLF +;
        '</AccessRequest>' + CRLF +;
        cRequest

      cHeaders := ;
        "Content-Length: " + NTrim(SLen(cRequest)) + CRLF +;
        "Accept-Language: en" + CRLF +;
        "Connection: Keep-Alive" + CRLF +;
        "Content-Type:application/x-www-form-urlencoded" + CRLF

      #IFDEF __DEBUG__
        MemoWrit(GetWorkDir()+"FullRequest.XML", cRequest)
        MemoWrit(GetWorkDir()+"Header.TXT", cHeaders)
      #ENDIF

      pBuffer := StringAlloc( cRequest )
      IF !oHttp:SendRequest(cHeaders, pBuffer, SLen(cRequest))
        cError := "SendRequest to URI: "+cURI+" failed"
        lError := TRUE
      ENDIF
      MemFree( pBuffer )

      IF !lError
        cXmlResult := oHttp:GetResponse()
        cXmlResult := StrTran( cXmlResult, _chr(10), CRLF )

        // XML-Fehler: <CODE> wird nicht erkannt
        cXmlResult := StrTran( cXmlResult, "<Code>", "<xCode>")
        cXmlResult := StrTran( cXmlResult, "</Code>", "</xCode>")

        pHandle := FCreate( cXmlResultsFile )
        FWrite( phandle, cXmlResult, SLen( cXmlResult))
        FClose( pHandle )

      ENDIF
    ENDIF
  ELSE
    cError := "Connect to host: "+cHost+" failed"
    lError := TRUE
  ENDIF
  oHttp := null_object

  RETURN lError
  ~"Onlyearly-"
```

```
METHOD UploadPLD2000( cPLD0200 AS STRING ) AS LOGIC PASCAL CLASS API_UPS

LOCAL cMSG_Teil1    AS STRING
LOCAL cMSG_Teil2    AS STRING
LOCAL cMSG_Teil3    AS STRING
LOCAL cRequest      AS STRING
LOCAL cHeaders      AS STRING
LOCAL cHtmlResult   AS STRING
LOCAL lSuccess      AS LOGIC
LOCAL lError        AS LOGIC
LOCAL pBuffer       AS PTR
LOCAL oHttp         AS cHttp

cMSG_Teil1 := "AppVersion=1.0&AcceptUPSLicenseAgreement=YES&ResponseType=application/x-ups-
pld&VersionNumber=V4R1&UserId="+SELF:cUserID+"&Password="+SELF:cPassword

cMSG_Teil2 := "Content-type: application/x-www-form-urlencoded"+CRLF
cMSG_Teil2 += "Content-length: " + NTrim(SLen(cMSG_Teil1))+CRLF+CRLF
cMSG_Teil2 += cMSG_Teil1

cMSG_Teil3 := "Content-type: application/x-ups-binary"+CRLF
cMSG_Teil3 += "Content-length: " + NTrim(SLen(cPLD0200))+CRLF+CRLF
cMSG_Teil3 += cPLD0200

cRequest   := "--BOUNDARY" + CRLF + cMSG_Teil2 + CRLF+CRLF + "--BOUNDARY" + CRLF + cMSG_Teil3
+ CRLF+CRLF + "--BOUNDARY--"+CRLF

cHeaders   := "Content-Type: multipart/mixed; boundary=BOUNDARY" + CRLF
cHeaders   += "Content-Length: " + NTrim(SLen(cRequest)) + CRLF + CRLF

// HTTP-Verbindung via Port 443 (SSL) initialisieren
oHttp := cHttp{ "UPS", 443 }

IF oHttp:ConnectRemote( cHost )

  IF ! oHttp:OpenRequest("POST", cURI
,INTERNET_FLAG_RELOAD+INTERNET_FLAG_SECURE+INTERNET_FLAG_IGNORE_CERT_CN_INVALID+INTERNET_FLAG_
IGNORE_CERT_DATE_INVALID )
    lError := TRUE
  ELSE
    pBuffer := StringAlloc( cRequest )

    lError := !oHttp:SendRequest(cHeaders, pBuffer, SLen(cRequest))
    IF lError
      MemFree( pBuffer )
    ELSE
      MemFree( pBuffer )

      cHtmlResult := oHttp:GetResponse()

      MemoWrit( GetWorkDir()+"HtmlResult.txt", cHtmlResult)

      // Antwort auswerten
      lSuccess := SELF:ParseResult( cHtmlResult )

    ENDIF
  ENDIF
ENDIF

RETURN lSuccess
```

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

2. Upload eines XML-Files auf einen HTTP-Server mit SSL

2.1. Grundsätzlicher Programmablauf

2.1.1. Aufgabe:

Es soll eine XML-Datei an folgende URL hochgeladen werden und die Antwort muss ausgewertet werden:

https://wwwcie.ups.com/ups.app/xml

Es funktioniert ganz ähnlich wie der ASCII-Upload. Anstelle eines ASCII-Files wird einfach ein XML-File hochgeladen.

2.2. Implementation

2.2.1. URL aufteilen

Die URL muss wiederum aufgeteilt werden:

https://wwwcie.ups.com/ups.app/xml

Der rote Teil gehört zum „ConnectRemote“, der blaue Teil zum „OpenRequest“

Unbedingt beachten: Nach dem ‚www’ DARF KEIN PUNKT folgen!

2.2.2. Aufbau der POST-Meldung (Request- und Hearder-String)

Der Request-String muss ebenfalls im XML-Format aufgebaut werden und sieht wie folgt aus:

```
<?xml version="1.0"?>CRLF
<AccessRequest xml:lang="en-US">CRLF
   <AccessLicenseNumber>8C0BS26C25AF184D</AccessLicenseNumber>CRLF
   <UserId>320001</UserId>CRLF
   <Password>xxxxxxxx</Password>CRLF
</AccessRequest>CRLF
<?xml version="1.0" encoding="ISO-8859-1"?><TimeInTransitRequest xml:lang="en-
US"><Request><TransactionReference><CustomerContext>00001292967</CustomerContext><XpciVersion>
1.0002</XpciVersion></TransactionReference><RequestAction>TimeInTransit</RequestAction></Reque
st><TransitFrom><AddressArtifactFormat><PoliticalDivision2>Bienne</PoliticalDivision2><Country
Code>CH</CountryCode><PostcodePrimaryLow>2501</PostcodePrimaryLow></AddressArtifactFormat></Tr
ansitFrom><TransitTo><AddressArtifactFormat><PoliticalDivision2>Madrid</PoliticalDivision2><Co
untryCode>ES</CountryCode><PostcodePrimaryLow>28028</PostcodePrimaryLow></AddressArtifactForma
t></TransitTo><ShipmentWeight><UnitOfMeasurement><Code>KGS</Code></UnitOfMeasurement><Weight>1
7</Weight></ShipmentWeight><TotalPackagesInShipment>2</TotalPackagesInShipment><InvoiceLineTot
al><CurrencyCode>USD</CurrencyCode><MonetaryValue>1200.00</MonetaryValue></InvoiceLineTotal><P
ickupDate>20071025</PickupDate></TimeInTransitRequest>CRLF
```

Dunkelgrün ist der zu übermittelnde XML-File. Zu beachten ist, dass darin KEINE CRLF vorkommen dürfen (ausser am Ende natürlich)!

Der dazugehörige Header:

```
Content-Length: 1179CRLF
Accept-Language: enCRLF
Connection: Keep-AliveCRLF
Content-Type:application/x-www-form-urlencodedCRLF
```

Grundsätzlich ist der Aufbau viel einfacher als ein ASCII-Upload

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

2.2.3. Serverantwort einlesen

Die Serverantwort sieht so aus:


```
<?xml version="1.0"?><TimeInTransitResponse><Response><TransactionReference>
<CustomerContext>00001292967</CustomerContext><XpciVersion>1.0002</XpciVersion></TransactionRe
ference><ResponseStatusCode>1</ResponseStatusCode><ResponseStatusDescription>Success</Response
StatusDescription></Response><TransitResponse><PickupDate>2007-10-25</PickupDate><TransitFrom>
<AddressArtifactFormat><PoliticalDivision2>BIENNE</PoliticalDivision2><PoliticalDivision1>BE</
PoliticalDivision1><Country>SWITZERLAND</Country><CountryCode>CH</CountryCode><PostcodePrimary
Low>2501</PostcodePrimaryLow></AddressArtifactFormat></TransitFrom><TransitTo><AddressArtifact
Format><PoliticalDivision2>MADRID</PoliticalDivision2><Country>SPAIN</Country><CountryCode>ES<
/CountryCode><PostcodePrimaryLow>28028</PostcodePrimaryLow></AddressArtifactFormat></TransitTo
><AutoDutyCode>01</AutoDutyCode><ShipmentWeight><UnitOfMeasurement><xCode>KGS</xCode></UnitOfM
easurement><Weight>17.0</Weight></ShipmentWeight><InvoiceLineTotal><CurrencyCode>USD</Currency
Code><MonetaryValue>1200.00</MonetaryValue></InvoiceLineTotal><Disclaimer>Services listed as
guaranteed are backed by a money-back guarantee for transportation charges only. See Terms and
Conditions in the Service Guide for details. Certain commodities and high value shipments may
require additional transit time for customs clearance.</Disclaimer><ServiceSummary><Service>
<xCode>01</xCode><Description>UPS Worldwide Express</Description></Service><Guaranteed>
<xCode>Y</xCode></Guaranteed><EstimatedArrival><BusinessTransitDays>1</BusinessTransitDays><Ti
me>10:30:00</Time><PickupDate>2007-10-25</PickupDate><PickupTime>18:00:00</PickupTime>
<HolidayCount>0</HolidayCount><DelayCount>0</DelayCount><Date>2007-10-26</Date><DayOfWeek>
FRI</DayOfWeek><TotalTransitDays>1</TotalTransitDays><CustomerCenterCutoff>17:00:00</CustomerC
enterCutoff><RestDays>0</RestDays></EstimatedArrival></ServiceSummary><ServiceSummary><Service
><xCode>28</xCode><Description>UPS Express Saver</Description></Service> <Guaranteed><xCode>Y
</xCode></Guaranteed><EstimatedArrival><BusinessTransitDays>1</BusinessTransitDays><Time>23:30
:00</Time><PickupDate>2007-10-25</PickupDate><PickupTime>18:00:00</PickupTime> <HolidayCount>0
</HolidayCount><DelayCount>0</DelayCount><Date>2007-10-26</Date> <DayOfWeek>FRI</DayOfWeek>
<TotalTransitDays>1</TotalTransitDays><CustomerCenterCutoff>17:00:00</CustomerCenterCutoff><Re
stDays>0</RestDays></EstimatedArrival></ServiceSummary><ServiceSummary><Service><xCode>68</xCo
de><Description>UPS Standard</Description></Service><Guaranteed><xCode>N</xCode></Guaranteed>
<EstimatedArrival><BusinessTransitDays>3</BusinessTransitDays><Time>23:30:00</Time><PickupDate
>2007-10-25</PickupDate><PickupTime>18:00:00</PickupTime><HolidayCount>0</HolidayCount>
<DelayCount>0</DelayCount><Date>2007-10-30</Date><DayOfWeek>TUE</DayOfWeek><TotalTransitDays>
5</TotalTransitDays><CustomerCenterCutoff>17:00:00</CustomerCenterCutoff><RestDays>2</RestDays
></EstimatedArrival></ServiceSummary><MaximumListSize>35</MaximumListSize></TransitResponse></
TimeInTransitResponse>CRLF
```

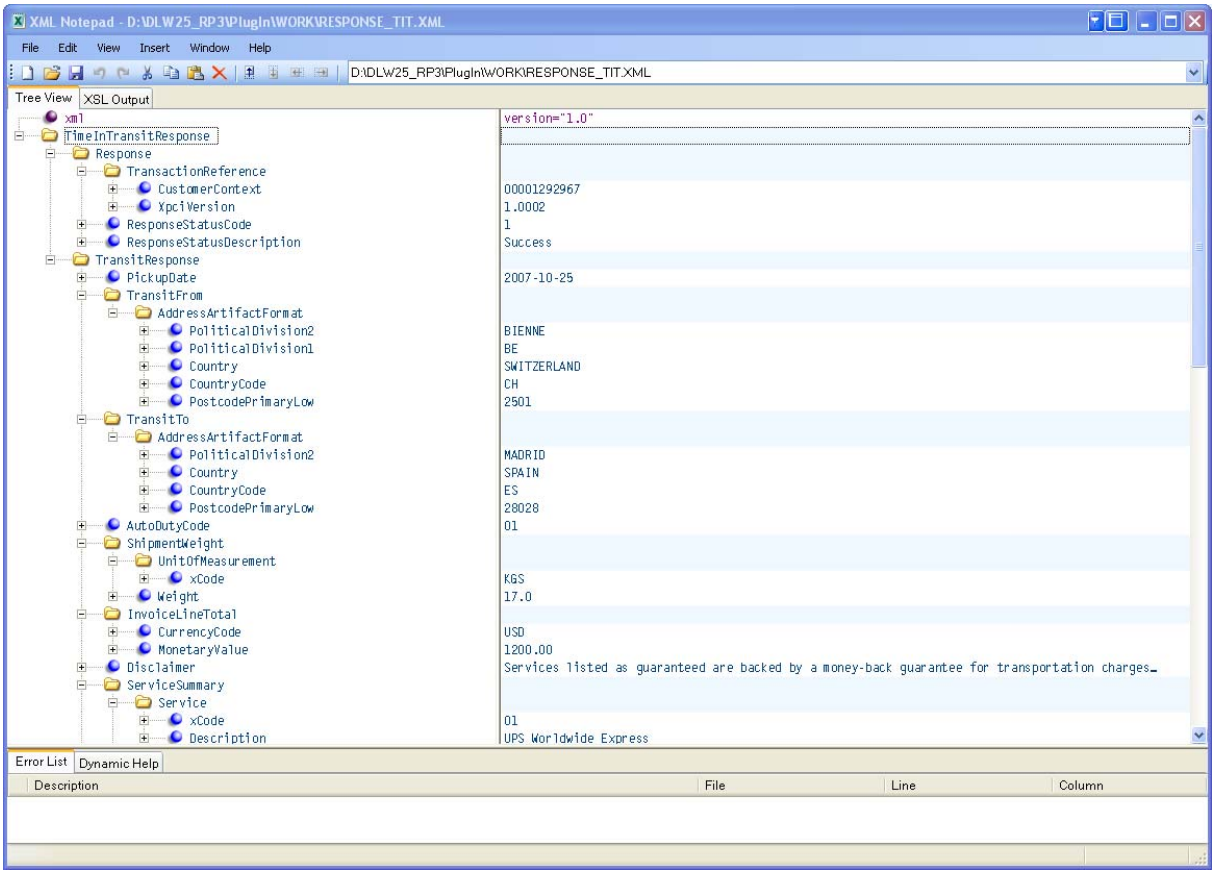

Das kann natürlich kein Mensch lesen.

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

2.2.4. XML anzeigen und bearbeiten

Ein recht gutes XML Anzeige- und Editierprogramm gibt's von Microsoft: das XMLNotepad
http://www.microsoft.com/downloads/details.aspx?familyid=72d6aa49-787d-4118-ba5f-4f30fe913628&displaylang=en

Damit sieht das Ganze schon wesentlich lesbarer aus:



2.3. Vollständiger Sourcecode

Der komplette Sourcecode einer XML-Abfrage:

```
METHOD UPSXMLRequestHttp( cRequest AS STRING, cXmlResultsFile AS STRING ) AS LOGIC PASCAL
CLASS API_UPS
  //p Führt eine XML-Anfrage an UPS durch
  //a cRequest \tab Anfrage als XML-Datei
  //a cXmlResultsFile \tab Pfad und Name. Die Serverantwort wird in diesem File abgespeichert
  //r lError \tab True, wenn ein Fehler aufgetreten ist.
        Der Fehlertext kann mit der Access:Methode cError abgerufen werden
  //d Der cRequest wird automatisch mit den Anmeldeinformationen ergänzt

  LOCAL oHttp         AS cHttp
  LOCAL pBuffer       AS PTR
  LOCAL cHeaders      AS STRING
  LOCAL cXmlResult    AS STRING       // Antwort der XML-Abfrage
  LOCAL pHandle       AS PTR
  LOCAL lError        AS LOGIC

  ~"Onlyearly-"
  oHttp := cHttp{ "UPS", 443 }
  IF oHttp:ConnectRemote( cHost )

    IF !oHttp:OpenRequest("POST",
cURI,INTERNET_FLAG_RELOAD+INTERNET_FLAG_SECURE+INTERNET_FLAG_IGNORE_CERT_CN_INVALID+INTERNET_F
LAG_IGNORE_CERT_DATE_INVALID )
      cError := "OpenRequest 'POST' to URI: "+cURI+" failed"
      lError := TRUE
    ELSE
      cRequest := ;
        '<?xml version="1.0"?>' + CRLF +;
        '<AccessRequest xml:lang="en-US">' + CRLF +;
        '   <AccessLicenseNumber>' + cXML_Key + '</AccessLicenseNumber>' + CRLF +;
        '   <UserId>' + cUserID + '</UserId>' + CRLF +;
        '   <Password>' + cPassword + '</Password>' + CRLF +;
        '</AccessRequest>' + CRLF +;
        cRequest

      cHeaders := ;
        "Content-Length: " + NTrim(SLen(cRequest)) + CRLF +;
        "Accept-Language: en" + CRLF +;
        "Connection: Keep-Alive" + CRLF +;
        "Content-Type:application/x-www-form-urlencoded" + CRLF

      #IFDEF __DEBUG__
        MemoWrit(GetWorkDir()+"FullRequest.XML", cRequest)
        MemoWrit(GetWorkDir()+"Header.TXT", cHeaders)
      #ENDIF

      pBuffer := StringAlloc( cRequest )
      IF !oHttp:SendRequest(cHeaders, pBuffer, SLen(cRequest))
        cError := "SendRequest to URI: "+cURI+" failed"
        lError := TRUE
      ENDIF
      MemFree( pBuffer )

      IF !lError
        cXmlResult := oHttp:GetResponse()
        cXmlResult := StrTran( cXmlResult, _chr(10), CRLF )

        // XML-Fehler: <CODE> wird nicht erkannt
        cXmlResult := StrTran( cXmlResult, "<Code>", "<xCode>")
        cXmlResult := StrTran( cXmlResult, "</Code>", "</xCode>")

        pHandle := FCreate( cXmlResultsFile )
        FWrite( phandle, cXmlResult, SLen( cXmlResult))
        FClose( pHandle )

      ENDIF
    ENDIF
  ELSE
    cError := "Connect to host: "+cHost+" failed"
    lError := TRUE
  ENDIF
  oHttp := null_object
  RETURN lError
  ~"Onlyearly-"
```

3. Der XML-Konverter

3.1. Bezugsquellen

Es gibt auf der Webseite von GrafX (http://support.cavo.com) einen XML-Konverter (Innovacts XMLParser) im Sourcecode.

Er hat den Vorteil, dass er kein ActivX benötigt. Leider ist er etwas buggy. Er unterscheidet z.B. keine Gross/Kleinschrift in den Tags, wodurch ein professioneller Einsatz verunmöglicht wird. Ebenso werden Tags, welche bestimmte Schlüsselwörter wie „<CODE>“ enthalten, ignoriert.

Eine halbwegs fehlerbereinigte Version kann von mir bezogen werden. Das Problem ist, dass die Tags intern als Symbol abgelegt werden, daher geht die Gross/Kleinschreibung natürlich verloren. Ein Umbau auf Stings anstelle Symbolen ist leider unmöglich. Ich habe eine kleine Tabelle eingebaut, welche die Gross/Kleinschreibung nachträglich wieder korrigiert. Für Neuentwicklungen würde ich diesen Parser daher nicht einsetzen.

Alternativen

Auf der Seite http://support.cavo.com gibt es noch einen weiteren Parser „XML Parser“, diesen habe ich noch nicht getestet.

Eine Variante mit OLE gibt’s von Norbert Kolb.

3.2. Auswerten der XML-Antwort

Zuerst müssen einige Macken des XML-Konverters ausgebügelt werden: (Siehe auch Sourcecode auf der vorigen Seite)

3.2.1. Zeilenumbruch einbauen

Zuerst wird ein Zeilenumbruch eingebaut, um die Lesbarkeit zu verbessern:

```
cXmlResult := StrTran( cXmlResult, _chr(10), CRLF )
```

3.2.2. Tag mit Schlüsselwort ersetzen

Der Konverter stolpert über den Tag <CODE> (!). Deshalb wird er ersetzt mit <xCode>

```
// XML-Fehler: <CODE> wird nicht erkannt
cXmlResult := StrTran( cXmlResult, "<Code>", "<xCode>")
cXmlResult := StrTran( cXmlResult, "</Code>", "</xCode>")
```

Nun kann der String an die Methode CheckXMLResponse übergeben werden.

© 2007 by Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald

3.3. Sourcecode der Auswertung der XML-Antwort

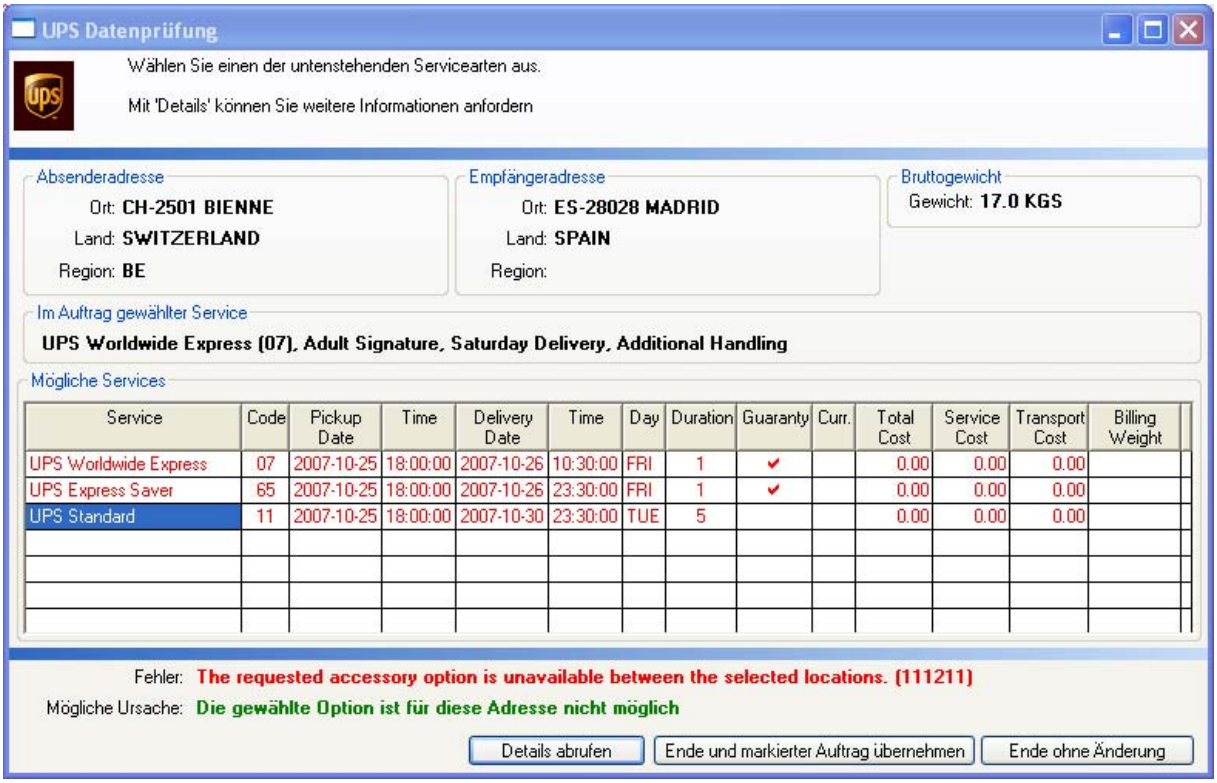
```
METHOD CheckXMLResponse( cXmlResultsFile AS STRING ) AS LOGIC PASCAL CLASS API_UPS

  LOCAL oParser           AS XMLParser2
  LOCAL ptrResponse       AS DWORD PTR
  LOCAL ptrError          AS DWORD PTR
  LOCAL ptrSubElement     AS DWORD PTR
  LOCAL ptrSubElement2    AS DWORD PTR
  LOCAL ptrSubElement3    AS DWORD PTR
  LOCAL ptrSubElement4    AS DWORD PTR
  LOCAL ptrRoot           AS DWORD PTR
  LOCAL lError            AS LOGIC

  LOCAL aStruct           AS ARRAY
  LOCAL nAdresse          AS DWORD
  LOCAL cTemp             AS STRING

  oParser := XMLParser2{}   // Create the object.

  // Tell the parser in the setup that you want to allocate 64K of mem for the file.
  oParser:SetUp(1)

  // Open the XML-file
  oParser:LoadFile ( cXmlResultsFile )

  // Get the root element.
  ptrRoot := oParser:RootElement        // Root = <TimeInTransitResponse>

  // Suche #Response
  ptrResponse := oParser:GetElementOnName ( ptrRoot , #Response)
  IF !Empty(ptrResponse)
    // #Response gefunden

    // Suche #TransactionReferenz
    ptrSubElement := oParser:GetElementOnName ( ptrResponse , #TransactionReference)
    IF !Empty(ptrSubElement)

      // #TransactionReferenz gefunden, Inhalt testen
      ptrSubElement := oParser:GetElementOnName ( ptrSubElement , #CustomerContext)
      IF !Empty(ptrSubElement)
        // Referenz gefunden
        cXMLReferenz := oParser:StringValue ( ptrSubElement )
      ENDIF
    ENDIF

    // Ist ein Fehler aufgetreten ?
    ptrSubElement := oParser:GetElementOnName ( ptrResponse , #ResponseStatusCode)
    IF !Empty(ptrSubElement)
      // #ResponseStatusCode gefunden, Inhalt testen

      IF oParser:DWordValue ( ptrSubElement ) = 1
        // Es ist kein Error aufgetreten

        // *****************
        // Antwort auswerten
        // *****************

        // Ist es ein TIT? Dann existiert ein <TransitResponse>
        ptrSubElement := oParser:GetElementOnName ( ptrRoot , #TransitResponse)
        IF !Empty(ptrSubElement)

          // Adressen einlesen
          aAddress := ArrayNew(2)

          FOR nAdresse := 1 UPTO 2
            IF nAdresse = 1
              ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement , #TransitFrom)
            ELSE
              ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement , #TransitTo)
            ENDIF
            ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement2 , #AddressArtifactFormat)

            IF !Empty(ptrSubElement2)
              aAddress[nAdresse] := ArrayNew( 5 )  // Land Kurz, Land, PLZ, Ort, Region
              // Land Kurz
              ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #CountryCode)
```

```
        aAddress[nAdresse][1] := oParser:StringValue( ptrSubElement3 )


        // Land
        ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #Country)
        aAddress[nAdresse][2] := oParser:StringValue( ptrSubElement3 )


        // PLZ
        ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #PostcodePrimaryLow)
        aAddress[nAdresse][3] := oParser:StringValue( ptrSubElement3 )


        // Ort
        ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #PoliticalDivision2)
        aAddress[nAdresse][4] := oParser:StringValue( ptrSubElement3 )


        // Region
        ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #PoliticalDivision1)
        aAddress[nAdresse][5] := oParser:StringValue( ptrSubElement3 )

      ENDIF
    NEXT

    // Gewicht
    ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement , #ShipmentWeight)
    ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #UnitOfMeasurement)
    ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement3 , #xCode)
    cGewicht := oParser:StringValue( ptrSubElement3 )

    ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #Weight)
    cGewicht := oParser:StringValue( ptrSubElement3 ) + " " + cGewicht

    // Alle <ServiceSummary> absuchen

    // Temporäre Datei zum Anzeigen erzeugen
    aStruct := {}
    AAdd(aStruct, { "Service",       "C", 25, 0 })
    AAdd(aStruct, { "Code",          "C",  2, 0 })
    AAdd(aStruct, { "Pickup_Date",   "C", 10, 0 })
    AAdd(aStruct, { "Pickup_Time",   "C",  8, 0 })
    AAdd(aStruct, { "Delivery_Date", "C", 10, 0 })
    AAdd(aStruct, { "Delivery_Time", "C",  8, 0 })
    AAdd(aStruct, { "Delivery_Day",  "C",  3, 0 })
    AAdd(aStruct, { "Duration",      "N",  3, 0 })
    AAdd(aStruct, { "Guaranty",      "L",  1, 0 })
    AAdd(aStruct, { "Currency",      "C",  3, 0 })
    AAdd(aStruct, { "TotalCost",     "N",  7, 2 })
    AAdd(aStruct, { "ServiceCost",   "N",  7, 2 })
    AAdd(aStruct, { "TransportCost", "N",  7, 2 })
    AAdd(aStruct, { "Bill_Weight",   "C", 10, 0 })
    AAdd(aStruct, { "Error",         "L",  1, 0 })
    AAdd(aStruct, { "ErrorCode",     "N",  6, 0 })
    AAdd(aStruct, { "cError",        "C",100, 0 })
    AAdd(aStruct, { "lChecked",      "L",  1, 0 })

    // Virtuelle Datenbank ArTemp anlegen
    oResponse := _bArrayServer{aStruct,,,, "oResponse"}

    // 1. Element suchen
    ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement , #ServiceSummary)
    IF !Empty(ptrSubElement2)
      DO WHILE TRUE

        ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #Service)

        oResponse:append()

        // Servicebeschreibung suchen  ('TiT' verwendet andere Codes als 'Rate')
        ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #xCode)
        cTemp := oParser:StringValue( ptrSubElement4 )
        cTemp := SubStr(Service_Codes, At(cTemp, TIT_Codes), 2)
        oResponse:FIELDPUT("CODE",  cTemp)

        // Servicebeschreibung suchen
        ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #Description)
        oResponse:FIELDPUT(#Service,  oParser:StringValue( ptrSubElement4 ))
```

```
          // Garantiert Y/N
          ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #Guaranteed)
          ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #xCode)
          oResponse:FIELDPUT(#Guaranty, iif(oParser:StringValue( ptrSubElement4 )="Y",TRUE,
FALSE))

          // Ankunftszeit
          ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #EstimatedArrival)
          ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #Time)
          oResponse:FIELDPUT(#Delivery_Time, oParser:StringValue( ptrSubElement4 ))

          // Ankunftsdatum
          ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #DATE)
          oResponse:FIELDPUT(#Delivery_Date, oParser:StringValue( ptrSubElement4 ))

          // Ankunftstag
          ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #DayOfWeek)
          oResponse:FIELDPUT(#Delivery_Day, oParser:StringValue( ptrSubElement4 ))

          // Pickupzeit
          ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #PickupTime)
          oResponse:FIELDPUT(#Pickup_Time, oParser:StringValue( ptrSubElement4 ))

          // Pickupdatum
          ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #PickupDate)
          oResponse:FIELDPUT(#Pickup_Date, oParser:StringValue( ptrSubElement4 ))

          // Laufzeit
          ptrSubElement4 := oParser:GetElementOnName ( ptrSubElement3 , #TotalTransitDays)
          oResponse:FIELDPUT(#Duration, Val(oParser:StringValue( ptrSubElement4 )))

          // Nächstes Element suchen
          ptrSubElement2 := oParser:GetNextSibling(ptrSubElement2)
          IF Empty(ptrSubElement2) .or. oParser:Name( ptrSubElement2 ) <> #ServiceSummary
            oResponse:commit()
            EXIT
          ENDIF
        ENDDO
      ENDIF
    ENDIF

    // Ist es ein Rate? Dann existiert ein <RatedShipment>
    aRates := {0,0,0}

    ptrSubElement := oParser:GetElementOnName ( ptrRoot , #RatedShipment)
    IF !Empty(ptrSubElement)

      // Billingweight
      ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement  , #BillingWeight)
      ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #UnitOfMeasurement)
      ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement3 , #xCode)
      cGewicht := oParser:StringValue( ptrSubElement3 )

      ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #Weight)
      cGewicht := oParser:StringValue( ptrSubElement3 ) + " " + cGewicht

      // Transportkosten
      ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement  , #TransportationCharges)
      ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #MonetaryValue)
      aRates[1] := Val(oParser:StringValue( ptrSubElement3 ))

      // Serviceoptionen-Kosten
      ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement  , #ServiceOptionsCharges)
      ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #MonetaryValue)
      aRates[2] := Val(oParser:StringValue( ptrSubElement3 ))

      // Totalkosten
      ptrSubElement2 := oParser:GetElementOnName ( ptrSubElement  , #TotalCharges)
      ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #CurrencyCode)
      cCurrency := oParser:StringValue( ptrSubElement3 )

      ptrSubElement3 := oParser:GetElementOnName ( ptrSubElement2 , #MonetaryValue)
      aRates[3] := Val(oParser:StringValue( ptrSubElement3 ))

    ENDIF

  ELSE
```

```
      // Es ist ein Error aufgetreten
      lError := TRUE

      // Error parsen
      ptrError := oParser:GetElementOnName ( ptrResponse , #Error)
      IF !Empty(ptrError)
        // Error-Element gefunden

        // Error-Code suchen
        ptrSubElement := oParser:GetElementOnName ( ptrError , #ErrorCode)
        IF !Empty(ptrSubElement)
          // Error-Code gefunden
          nXMLError := oParser:DWordValue ( ptrSubElement )
        ENDIF

        // Error-Beschreibung suchen
        ptrSubElement := oParser:GetElementOnName ( ptrError , #ErrorDescription)
        IF !Empty(ptrSubElement)
          // Error-Beschreibung gefunden
          cXMLError := oParser:StringValue ( ptrSubElement )
        ENDIF

      ELSE
        cXMLError := "No error-description found!"
      ENDIF

    ENDIF

  ENDIF
ENDIF

RETURN lError
```

Das Resultat ist eine RAM-DBF, welche optisch ansprechend dargestellt werden kann:



Der Autor kann erreicht werden unter spamspam@bluewin.ch