byron

Byron Informatik AG
Eisengasse 6
CH-4051 Basel
Tel. +41 (0)61 690 96 00
Fax +41 (0)61 690 96 09

byron@byron.ch
www.byron.ch

Byron/BIS – Technische Produktinformation
Script Beschreibung für BIS*XDS*

Inhaltsverzeichnis

1 Einführung

In XDS gibt es eine Vielzahl von Aufgaben (Mailversand, Überwachen von Dateien…), welche jeweils parametrisiert sind. Ein anderer Ansatz bietet die hier beschriebene Script-Methode. Ein Script besteht aus einzelnen Grundkonstrukten, welche zur Lösung einer Aufgabe kombiniert werden können.

Im Folgenden werden die einzelnen Grundkonstrukte (Script-Elemente) und die Anwendung dieser für eine Aufgabe beschrieben.

Das Prinzip des Script besteht darin, dass Objekte die einzelnen Element (Anweisungen) durchlaufen und damit etwas auslösen. Dieses Prinzip ist also mit der Filter-Navigation vergleichbar, mit dem Unterschied, dass die Effekte des Durchlaufens wesentlich sind und nicht die Resultatmenge.

Das Prinzip an einem Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="fn" Value="=EXTRACTNAME(fpn)" />
  </Variables>
  <Script>
    <!-- kein Objekt 1) -->
    <Navigation>
      START INSTANCES Doc_File
      VALUE Doc_Path '*.drw'
    </Navigation>
    <!-- alle Condor Pläne -->
    <ForEachObject>
      <!-- Dateipfad in eine Variable fpn schreiben -->
      <SetVar BisAttribute="Doc_Path" Name="fpn" />
      <!-- Variable fn in Log des XDS-Tools schreiben -->
      <LogEntry Value="'Plan Name: '+ fn" />
    </ForEachObject>
  </Script>
</Defs>
```

Das Script ermöglicht mit FetchError / OnError eine Transaktionssteuerung und Fehlerbehandlung. Wenn diese nicht angewendet wird läuft das Script in einer Transaktion. Wenn notwendig kann dies eine Schreibtransaktion sein, sonst eine Lesetransaktion. Wenn eine an einer Stelle eine Schreibtransaktion benötigt wird und andererseits ein Benutzerdialog verwendet wird, dann wird eine explizite Transaktionssteuerung reklamiert.

Achtung: Wird FetchError / OnError verwendet, dann liegt eine explizite Transaktionssteuerung vor und alle Skriptelemente sollten in FetchError / OnError geklammert werden.

In Scripts werden häufig Expressions verwendet. Darin werden zu den Standardfunktionen noch die drei Funktionen `PARAM(name : string)`, `USER` und `ATTRIBUTE(atrName : string)` unterstützt. Letztere erlaubt eine kleine Abkürzung:

Statt

```
<SetVar Name="vId" BisAttribute="'bisB_ObjectImage "/>
<XMLattribute Name="'id'" Value="SUBSTR(vId,2,36)" />
```

mit `ATTRIBUTE;`

```
<XMLattribute Name="'id'"
              Value="SUBSTR(ATTRIBUTE('bisB_ObjectImage'),2,36)" />
```

1.1 Funktionen in Ausdrücken

An vielen Stellen sind Ausdrücke (Expressions) erlaubt. Es ist ein Unterschied, ob in einem XML-Attribut ein Text oder eine Expression erwartet wird. Wenn ein Text erwartet wird kann man Tisch schreiben, wenn es eine Expression ist, dann muss man 'Tisch' schreiben, sonst wird die Variable Tisch eingesetzt, welche dann leer ist, da sie undefiniert ist. Zusätzlich zu den in support.byron.ch/downloads/dokumente/ExpressionsForVariablesAndValues.pdf definierten Funktionen sind im XDS Konzept zusätzlich noch folgende definiert:

`PARAM(name : text) : text`
Gibt den Wert des Globalen-Parameter von name. Bsp.:

```
<LogEntry Options="always" Value="PARAM('ExtDocPath_Reservation')"/>
```
PARAM kann den Globalen-Parameter nicht auswerten, sondern nur lesen. Das heisst, wenn im Wert des Globalen-Parameter weitere Parameter oder Umgebungsvariablen vorkommen, werden diese nicht aufgelöst.

`USER(attrib : text) : text`
Attribut des aktuellen Benutzer, wenn attrib nicht definiert ist, dann wird ‚Name' angenommen.

`ATTRIBUTE(attrib : text) : text`
Evaluiert das Attribut auf dem 1. Objekt. Wenn attrib undefiniert ist wird Object_Display_Kontext angenommen.

`EVAL(name : text) : text`
Wertet Globale-Parameter und Umgebungsvariablen in name aus. Bsp.:

```
<LogEntry Options="always"
          Value="EVAL('Der ExtDocPath: %ExtDocPath_Reservation%')"/>
```

Mit "%%" können sowohl Umgebungsvariabeln als auch globale Parameter beliebig in den String einbaut werden. Da sie ausgewertet und nicht nur gelesen werden, können die Werte weitere globale Parameter oder Umgebungsvariabeln beinhalten. Ein globaler Parameter kann sogar eine Umgebungsvariabel verwenden.

`CODED(name : text) : text`
Erstellt einen codierten Parameter, welcher Datenbank, Benutzer, Passwort und name enthält. Letzteres wird von BisXDS.exe als Abgleichname verwendet. Dieser Code kann als Commandlineparameter von Zusatzprogrammen von Byron für den Zugriff auf die Datenbank genutzt werden. Für das Ausführen eines anderen XDS-Script (ExecXDS) muss diese Funktion nicht verwendet werden, der Parameter wird schon von ExecXDS eingesetzt. Wenn XDS nicht über einen codierten Parameter gestartet wurde, dann ergibt diese Funktion einen Leertext, sonst wird der Code inklusive, d.h. innerhalb von " zurückgegeben.

1.2 DEFINE / USE (ab v4.6.1)

Ein `<DEFINE/>`-Element hat einen Namen. Mit einem `<USE/>`-Element kann ein `<DEFINE/>` über den Namen referenziert werden, d.h. alle Elemente des `<DEFINE/>` werden an der Stelle des `<USE/>` eingefügt. Damit können gleiche Sequenzen von Elementen nur einmal geschrieben werden.

Beispiel:

anstatt

```
<aa/>
```

```
<bb>
  <b1/>
  <b2/>
</bb>
<cc/>
<aa/>
<bb>
  <b1/>
  <b2/>
</bb>
```

schreibt man kürzer:

```
<DEFINE Name="ab">
  <aa/>
  <bb>
    <b1/>
    <b2/>
  </bb>
</DEFINE>
<USE Name="ab"/>
<cc/>
<USE Name="ab"/>
```

Das `<DEFINE/>`-Element muss sich auf oberster Stufe befinden, also unterhalb von `<Defs/>`.

Diese Ersetzung erfolgt vor dem eigentlichen Lesen der Definition (Preprocess) und gilt für das gesamte XDS, nicht nur für den Script Teil.

2 Script Elemente

Der Sinn vieler Elemente ergibt sich erst durch ihre Anwendung siehe Anwendung

Das Wurzelelement ist `<Script/>`, es befindet sich unterhalb der Dokumentwurzel, d.h. auf gleichem Level wie `<Variables/>`.

Beispiel eines leeren Rumpfes:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs Trace="off"> <!-- vgl. XDS_IntroductionReference Kapitel 3.1 -->
  <Variables>
  </Variables>
  <Script>
     <!-- hier kommen die Script Befehle -->
  </Script>
</Defs>
```

2.1 AddLookup / ClearLookup / GetLookup / ForEachLookup

Eine Lookup-Tabelle besitzt einen Namen. Mit `<ClearLookup/>` wird eine Lookup-Tabelle gelöscht. Mit `<AddLookup/>` wird der Tabelle ein Eintrag hinzugefügt, mit `<GetLookup/>` wird ein Eintrag abgefragt. Wenn zwei Datenmengen miteinander verglichen werden, dann kann eine Datenmenge in die Lookup-Tabelle eingefügt werden. Bei der Abfrage der zweiten Datenmenge kann die Lookup-Tabelle verwendet werden, um zu testen ob es diesen Wert in der anderen Menge gibt. Dies ist wesentlich effizienter als zwei verschachtelte Schleifen.

```
<ClearLookup Name="myLookup" />
<AddLookup   Name="myLookup" KeyValue="kv" StoreValue="val" />
<GetLookup   Name="myLookup" KeyValue="kv" ToVar="vv" />
```

`KeyValue` und `StoreValue` sind Expression vom Typ Text. Default von `StoreValue` ist derselbe Wert wie `KeyValue`.

`ToVar` bezeichnet eine Variable, in welcher der Wert (`StoreValue`) von kv gespeichert wird, oder NULL wenn keiner definiert ist.

ab V4.9.4 `<AddLookup/>` mit `Replace` <Boolean>
`<AddLookup/>` kann kann auch Werte von Keys überschreiben, wenn `Replace="true"` gesetzt wird, sonst gibt gibt `<AddLookup/>` einen Fehler wenn es den Key bereits gibt. Diese Flag ist obsolet und sollte durch `Mode="replace"` formuliert werden.

ab V4.9.8 `<AddLookup/>` mit `Mode` mögliche Werte sind: `'raiseError', 'replace', 'sum', 'avg', 'min', 'max'` damit wird das Verhalten bei gleichen Schlüsselwerten gesteuert. Bei `raiseError` (default) ergeben gleiche Schlüssel einen Fehler. Mit `replace` wird der Wert jeweils ersetzt (‚der Letzte gewinnt'). Bei den übrigen Modi muss der Wert numerisch sein damit Summe, Mittelwert, Minimum oder Maximum gebildet werden kann.

Folgendes Beispiel liest Daten mit der ersten Spalte ein Hierarchie Level fEbene als Integer und in der 2. Spalte fFacId als Schlüssel.

```
<OpenFile Mode="read" FileName="fn" RecVar="rec" Sep="#9" Quote="#34">
  <SetVar Name="fRecNr" Expression="fRecNr + 1" />
  <IF Condition="fRecNr &gt; 1">
    <SetVar Name="fEbene"  Expression="FIELD(rec, 0)" />
    <SetVar Name="fFacId"  Expression="FIELD(rec, 1)" />
```

```
    <SetVar Name="fdatStr" Expression="FIELD(rec, 5)" />
    <Navigation>
      START VALUE bisRW_ID = DATAOF fFacId
    </Navigation>
    <SetVar Name="objSrgt" BisAttribute="bisB_ObjectImage" />
    <IF Condition="STRLEN(objSrgt) = 0">
      <!-- den gibts noch nicht neu erzeugen -->
      <GetLookup Name="levs"
                 KeyValue="FORMAT(INT(fEbene)-1,'%d')" ToVar="par" />
      <IF Condition="STRLEN(par) = 0">
        <Navigation>
          CREATE 1 'bisRW_RegISElement' (RECALL vKatalog)
          MODIFY bisRW_ID DATAOF fFacId
        </Navigation>
      <ELSE/>
        <Navigation>
          CREATE 1 'bisRW_RegISElement' (START DATAOF par)
          MODIFY bisRW_ID DATAOF fFacId
        </Navigation>
      </IF>
      <SetVar Name="objSrgt" BisAttribute="bisB_ObjectImage" />
    </IF>
    <AddLookup Name="levs" Replace="true"
               KeyValue="fEbene" StoreValue="objSrgt" />
    <Navigation>
      MODIFY Name         '=FIELD(rec, 2)'
      MODIFY bisRW_Remark '=FIELD(rec, 3)'
      MODIFY bisRW_Source '=FIELD(rec, 4)'
    </Navigation>
  </IF>
</OpenFile>
```

Die Lookup kann auch zum gruppieren von Werten verwendet werden.

In `<AddLookup>` gibt man im `Mode` die Operation an, welche bei gleichen Werten angewendet werden soll. Anschliessend kann mit `<ForEachLookup>` die Daten wieder ausgelesen werden.

```
<ForEachLookup Name="myLookup" KeyVar="kv" ValueVar="vv"/>
```

`Name` und `Value` der einzelnen Einträge werden in die mit `KeyVar` und `ValueVar` bezeichneten Variablen abgelegt.

Hier ein Beispiel welches die Kosten von gleichen Records aufsummiert und dafür nur eine Zeile in eine Datei schreibt.

```
<Navigation>
  [
  OR
    LOOP (VIA Object_To_Children)
  ]
  CLASS ABF_Rechnungsposition
</Navigation>
<ForEachObject>
  <SetVar Name="vVon"
          Attribute="ABF_Abrechnungszeit" Format="DDMMYYYY" />
  <SetVar Name="vNu"
```

```
          Attribute="ABF_RechnungZuMieter%bisC_CostCenterId" />
  <SetVar Name="vKos" Attribute="ABF_Gesamtpreis" /> <!-- als float -->
  <SetVar Name="lin" Expression="vVon +#9+ vNu" />
  <AddLookup KeyValue="lin" StoreValue="vKos" Mode="sum" />
</ForEachObject>
<OpenFile Mode="create" FileName="fn" encoding="ISO-8859-1">
  <ForEachLookup KeyVar="lin" ValueVar="kos">
    <SetVar Name="rec" Expression="lin +#9+ FORMAT(kos, '%.2f')" />
    <Next Variable="rec" />
  </ForEachLookup>
</OpenFile>
```

2.2 ADOconnection / ADO… (ab v4.5)

Eine ADOconnection stellt eine Verbindung zu einer ADO-Datenquelle her. Innerhalb der ADOconnection kann man auf die Daten zugreifen.

Es gibt ein Beispiel mit einem LDAP Zugriff: Personen Import mit LDAP (ADO)

Der Datenzugriff erfolgt über `<ADOgetField>` das Feld wird über sein Namen oder die Nummer (0..) angesprochen: `GetByName="'ID'"` oder `GetByNumber="0"`. Mit dem Attribut Info wird angegeben, welche Information des Feldes gelesen werden soll: Wert, Name, Wert als Text…

Mit ADOnext kann auf den nächsten Record zugegriffen werden.

Zugriffsmuster zum lesen aller Records:

```
<ADOconnection Id="ado">
  <ADOfirst Id="ado" Variable="ok" />
  <WHILE Condition="ok">
     <ADOgetField Id="ado" GetByNumber="0" Variable="vId" />
     <!-- mit vId etwas Anfangen, auf weitere Felder zugreifen -->
     <ADOnext Id="ado" Variable="ok" />
 </WHILE>
</ADOconnection>
```

Manipulationen können gemacht werden, indem man mit ADOsetField die Felder beschreibt. Mit <ADOaddnew> kann ein neuer erstellt werden, der dann beschrieben. Je nachdem ob man ADOupdate anwedet wird ein anderer Lock-Mechanismus verwendet.

```
<ADOaddnew Id="ado" />
<ADOsetField Id="ado" GetByName="'ID'" Value="'BIT.1.2/x00101'" />
<ADOsetField Id="ado" GetByName="'TCPIP'" Value="'167.16.16.16'" />
<ADOsetField Id="ado" GetByName="'Betriebssystem'" Value="W/128'" />
<ADOsetField Id="ado" GetByName="'Inbetrieb'" Value="NOW()" />
<ADOsetField Id="ado" GetByName="'Zustand'"   Value="3" />
<ADOsetField Id="ado" GetByName="'Genutzt'"   Value="1" />
<ADOupdate Id="ado" />
```

2.2.1 ADOconnection

Attribut `Id` : text
andere ADO Elemente beziehen sich mit dieser Id auf die ADOconnection.

Attribut `ConnectionString` : expression
Definiert den ConnectionString als expression, also typischerweise den Namen einer Variablen.

Beispiel Excel 2007:
Provider=Microsoft.ACE.OLEDB.12.0;Excel 12.0;Database=C:\dir\data.xlsx

Beispiel Excel 2019 / Office 365:
Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\dir\data.xlsx;Extended Properties="Excel 12.0 Xml;HDR=YES";
(Unter Office365 "Klick und Los"-Installation (Fehlermeldung «Provider nicht gefunden») ist die Installation des Microsoft Access Database Engine 2016 Redistributable erforderlich. Installation mit dem Parameter /quiet, um die Fehlermeldung bez. 32/64Bit zu umgehen.)

Attribut `Select` : expression
Wenn man direkt ein Select statement angib muss dieses in , , geschrieben werden, das es sich um eine Expression handelt.

Attribute `FieldCountVariable` : string (optional)
Schreibt die Anzahl Felder in die Variable mit diesem Namen.

Attribute `RecordNumberVariable` : string (optional)
Schreibt die Anzahl Records in die Variable mit diesem Namen.

2.2.2 ADOgetField

Attribut `Id` : text
vgl. ADOconnection.

Attribut `GetByNumber` : integer / `GetByName` : text
Eines von beiden muss definiert sein und bestimmt damit das Feld. Nachfolgend gibt es ein Beispiel, welches alle Namen der Felder protokolliert.

Attribut `Variable` : text
Definiert den Namen der variable, in welchen der Wert geschrieben wird.

Attribut `Info` : (value | type | text) default = „value"
Definiert welche Information gelsene werden soll:
value: der Wert typisiert, so wie er im ADO ist
text: der Wert als Text
type: eine Nummer, die dem Datentyp des Feldes entspricht
name: der Name des Feldes. (Der wird in GetByName verwendet)

2.2.3 Beispiel

Alle Felder rausschreiben:

```
<Variables>
  <Var Name="fn">=OSENV('allusersprofile')
    + '\ByronBIS\DemoDb\Computer.mdb'
  </Var>
  <Var Name="cns">='Provider=Microsoft.Jet.OLEDB.4.0;'
    + 'Data Source=' + fn + ';Persist Security Info=False'
  </Var>
</Variables>
<Script>
  <LogEntry Value="'lese: ' + fn" />
  <ADOconnection Id="ado"
                 ConnectionString="cns"
                 Select="'SELECT * FROM Computer'"
                 FieldCountVariable="adoFieldCount"
                 RecordNumberVariable="adoRecordNumber">
```

```
    <LogEntry Value="'Anzahl Felder : ' + FORMAT(adoFieldCount, '%d')" />
    <LogEntry Value="'Anzahl Records: '+ FORMAT(adoRecordNumber, '%d')" />
    <FOR Variable="i" To="adoFieldCount-1">
      <ADOgetField Id="ado" GetByNumber="i"
                   Variable="n" Info="name" />
      <ADOgetField Id="ado" GetByNumber="i"
                   Variable="v" Info="type" />
      <LogEntry Value="'Field name: ' +n+ ' type: '+ FORMAT(v, '%d')" />
    </FOR>
  </ADOconnection>
</Script>
```

Folgendes Beispiel verwendet dieselben Variablen wir vorhergehendes Beispiel, schreib aber alle Werte von 2 Spalten ins Log. ‚RecordNumberVariable' ist ein aufwendiger Zugriff, desshalb empfiehlt sich das nachfolgende Zugriffsmuster:

```
<Script>
  <LogEntry Value="'lese: ' + fn"/>
  <ADOconnection   Id="ado"
                   ConnectionString="cns"
                   Select="'SELECT * FROM Computer'">
    <ADOfirst Id="ado"  Variable="ok" />
    <WHILE Condition="ok">
      <ADOgetField Id="ado" GetByNumber="0" Variable="vId" />
      <ADOgetField Id="ado" GetByName="'TCPIP'" Variable="vIp" />
       <LogEntry Value="'Value: ' + vId + '  IP: ' + vIp" />
       <ADOnext Id="ado" Variable="ok" />
    </WHILE>
  </ADOconnection>
</Script>
```

Das Beispiel ausgebaut in dem der Record mit dem Wert ‚PX7, im Feld ‚0' die IP-Adresse geändert wird:

```
<WHILE Condition="ok">
  <ADOgetField Id="ado" GetByNumber="0" Variable="vId" />
  <IF Condition="vId = 'PX7'">
     <SetVar Name="vIp" Expression="'167.18.16.50'" />
     <ADOsetField Id="ado" GetByName="'TCPIP'" Value="vIp" />
   </IF>
   <ADOnext Id="ado" Variable="ok" />
</WHILE>
```

Folgendes Beispiel macht eine Verbindung mit Excel:

```
<Variables>
  <Var Name="fn" Value="='H:\Kunden\Telefonimport\Telefonbuch.xls'" />
  <Var Name="cns">='Provider=Microsoft.Jet.OLEDB.4.0;'
       + 'Data Source=' + fn
       + ';Extended Properties=Excel 8.0;Persist Security Info=False'
  </Var>
</Variables>
<Script>
   <ADOconnection Id="ado" ConnectionString="cns"
       Select="'SELECT * FROM [Person$] ORDER BY Name'">
```

```
    <!-- hier kommt der ADO code -->
  </ADOconnection>
</Script>
```

Folgendes Beispiel verwendet LDAP. Dieser Provider enötigt noch einen Parameter ‚Page size' mit Wert 10000:

```
<Variables>
  <Var Name="cns" Value="='Provider=ADsDSOObject; Integrated Security=SSPI'"/>
  <Var Name="sel">='select cn, sn, samaccountname '
    + 'from ''LDAP://byron'' where objectclass=''user'' and sn>''a'' '
  </Var>
</Variables>
<Script>
  <LogEntry Value="'lese: ' + fn" />
  <ADOconnection Id="ado" ConnectionString="cns" Select="sel">
    <ADOconnectionParameter  Name="Page size" Value="1000" />
    <ADOfirst Id="ado"  Variable="ok"/>
    <WHILE Condition="ok">
      <ADOgetField Id="ado" GetByNumber="0" Variable="vCn" />
      <ADOgetField Id="ado" GetByName="'sn'" Variable="vSn" />
      <ADOgetField Id="ado" GetByName="'samaccountname'"
                   Variable="vSac" />
      <Navigation>
        START VALUE bisB_Lastname = DATAOF vSn
        [
          MODIFY Name  vCn
        ELSIF
          CREATE 1 Person MODIFY bisB_Lastname DATAOF vSn
        ]
      </Navigation>
      <LogEntry Value="'CN: ' + vCn + ' SN: ' + vSn + ' name ' + vSac" />
      <ADOnext Id="ado" Variable="ok" />
    </WHILE>
  </ADOconnection>
</Script>
```

2.3 CadExport / SourceFile / TargetFile / TextGraphic

CadExport erzeugt eine CAD-Datei (TargetFile), welche aus einem Muster (SourceFile) besteht und auf einem definierten Layer Text Elemente (TextGraphic) an der Position der Objekte generiert.

Die Dateien werden durch das Attribut Doc_Path des jeweiligen Objektes definiert.

CadExport
definiert die allgemeinen Einstellungen und beinhaltet die übrigen Elemente. Die Attribute definieren Einstellungen, welche die generierten Texte nutzen (vgl. TextGraphic):

Layer:	Definiert den Layer auf welchen die Texte (vgl. TextGraphic) gesetzt werden.
FontName:	Definiert den Font, welcher für den Text angewendet wird.
FontSize:	Definiert die Fontgrösse der Texte.

TargetFile
kein Attribut. Datei = Doc_Path vom aktuellen Objekt

SourceFile
kein Attribut. Datei = Doc_Path vom aktuellen Objekt

UseTransformation
kein Attribut Transformation wird aus bisG_Null* vom aktuellen Objekt (Stockwerk) gelesen.

Muster

```
<Navigation>
  START '{BB742B6D-8FF5-463D-AB04-A2360810B487}'
</Navigation>
 <CadExport Layer="G-EINBAUTEN"
            FontName="Verdana"
            FontSize="40">
   <UseTransformation/>
   <Navigation>
     VIA Doc_Documents VALUE Doc_Path '*exp.drw'
   </Navigation>
   <TargetFile/>
   <Navigation>
     START '{BB742B6D-8FF5-463D-AB04-A2360810B487}'
     VIA Doc_Documents VALUE Doc_Path '*orig.drw'
   </Navigation>
    <SourceFile/>
   <Navigation>
     START '{BB742B6D-8FF5-463D-AB04-A2360810B487}'
     LOOP (VIA Object_To_Children)  VIA Room_To_Component
   </Navigation>
   <ForEachObject>
     <SetVar Name="key" Attribute="Object_Display_Typed"/>
     <TextGraphic Value="key"/>
   </ForEachObject>
  </CadExport>
```

2.4 CadImport / ForEachCADobject /-block /-text

Mit CadImport kann eine CAD Datei gelesen werden. Innerhalb der CAD Import wird dann je nach Objekt ForEachCADblock, ForEachCADtext, oder ForEachCADobject ‚aufgerufen'.

`ForEachCADtext` kennt wie die anderen Elemente (ForEachCADobject, ForEachCADblock) die Attribute:

`Level`: Integer; Verschachtelungstiefe.

`Model`: Integer; Gruppe (Hintergrund, Vordergrund. Bzw. Modellbereich, Papierbereich..

Zusätzlich für `ForEachCADtext` gibt es:

`Text`: Text; enthält den Text (welcher in der Zeichnung sichtbar ist).

Wenn es sich beim Text um ein CAD-Attribut oder ein EED handelt, dann gibt es noch zusätzlich:

`BlockName`: Text; Bezeichnung des referenzierten Blockes.

`AttrTxt`: Text; Bezeichnung des Attributes.

`x`: Float; x-Koordinate des referenzierten Blockes (bzw. Einfügepunkt).

y: Float; y-Koordinate des referenzierten Blockes (bzw. Einfügepunkt).

sx: Float; x-Skalierung des referenzierten Blockes.

sy: Float; y-Skalierung des referenzierten Blockes.

rot: Float; Drehung des referenzierten Blockes.

```
<Variables DateLiterals="no">
  <Var Name="dir"
       Value="C:\Dokumente und Einstellungen\All Users\ByronBIS\DemoDb\CAD-Files\"
/>
  <Var Name="ftxt" Value="=dir + EXTRACTNAME(fpn) + '.txt'" />
</Variables>

<Script>
  <ForEachFile Dir="dir" FilePattern="'A20-01-*.dwg'" ToVar="fpn">
    <LogEntry Value="'Lese Datei: ' + fpn"/>
    <OpenFile Id="out" Mode="create" FileName="ftxt">
      <FetchError BisUpdate="yes">
        <CadImport FileName="fpn">
          <ForEachCADblock Name="blrName" Level="lev" x="x" y="y" Model="mdl" Re-
cursive="mdl = 1">
            <LogEntry Value="'Block:  ' + blrName + FORMAT(x, ' x %f') + FOR-
MAT(lev, ' level %d')" />
        <SetVar Name="rec" Expression="FORMAT(mdl, '%d') + #9 + FORMAT(lev, '%d') +
#9 + 'block' + #9 + blrName + #9 + FORMAT(x, '%f') + #9 + FORMAT(y, '%f')" />
            <Next Id="out" Variable="rec" />
          </ForEachCADblock>
          <ForEachCADtext Text="Text" Level="lev" Model="mdl">
            <LogEntry Value="'Text:  ' + Text + FORMAT(lev, ' level %d')" />
        <SetVar Name="rec" Expression="FORMAT(mdl, '%d') + #9 + FORMAT(lev, '%d') +
#9 + 'text' + #9 + Text" />
            <Next Id="out" Variable="rec" />
          </ForEachCADtext>
          <ForEachCADobject Type="oType" Level="lev" Model="mdl">
            <SetVar Name="rec" Expression=
"FORMAT(mdl, '%d') + #9 + FORMAT(lev, '%d') + #9 + FORMAT(oType, '%d')" />
            <Next Id="out" Variable="rec" />
          </ForEachCADobject>
        </CadImport>
      <OnError Abort="yes"/>
        <LogEntry Value="'Fehler in Datei: ' + fpn" />
      </FetchError>
    </OpenFile>
  </ForEachFile>
</Script>
```

2.5 Dir (ab v4.5.1)

Mit Dir können Operationen auf Verzeichnisse ausgeführt werden. Es gibt dazu folgende Attribute:

Name: Verzeichnis als Ausdruck.

Target: Zieldatei als Ausdruck, dies wird beim Kopieren benötigt.

Do ein Wert von: Create, Copy, Delete (Copy ab 5.3.1)

```
<Script>
  <Dir Name="'C:\Temp\Test'" Do="Create"/>
  <Dir Name="'C:\Temp\Test'" Do="Delete"/>
  <Dir Name="EVAL('%tmp%') + '\kacheln' " Target="H:\Gis\Tiles' Do="Copy"/>
</Script>
```

2.6 ExcelToText

Damit kann eine Excel Datei in eine Tab getrennte Textdatei konvertiert werden.

Die beiden Attribute Excel und Text sind Expression für die Dateien. Wenn es die Zieldatei bereits gibt es einen Bestätigungsdialog für das Überschreiben. Um das zu verhindern kann die Datei zuerst gelöscht werden.

Beispiel

```
<Variables>
  <Var Name="ftxt" Value="=OSENV('TEMP') + '\excelData.txt'"/>
</Variables>

<Script>
  <FileDialog Open="yes"
              ResultVar="fexcel"
              Title="Excel Konvertieren"
              Filter="'Excel (*.xls)|*.xls|Alles (*.*)|*.*'"
              DefaultExt="xls" />
  <IF Condition="REGEXP(fexcel, '\.xls')">
    <File Do="delete" Name="ftxt" />
    <ExcelToText Excel="fexcel" Text="ftxt" />
  </IF>
</Script>
```

Attribute von ExcelToText:

`Excel : Expression Text.` Excel Dateiname (Quelle).

`Text : Expression Text.` Text Dateiname. Diese Datei wird neu erstellt.

`SheetIndex : integer Expression` ab V4.10.2
Definiert welches Blatt gespeichert wird. Das erste `Blatt` hat den Index 0. Wenn der Wert weggelassen wird, dann wird das aktuelle Blatt verwendet. Vgl. auch Beispiel untenn

`SaveAsCode : Integer` Default ="20" Workbook SaveAs XlFileFormat:
xlTextWindows = 20
xlUnicodeText = 42

Siehe auch MS-EXCEL – Datei exportieren

Beispiel über alle Tabellen iterieren ab V5.0.2 :

```
<FOR Variable="vSheetIndex" From="0" To="99">
```

```
  <ExcelToText Excel="vOrigFile" Text="vTempFile"
               SheetIndex="vSheetIndex" />
  <IF Condition="FILESIZE(vTempFile) > 0">
    <OpenFile Mode="read" FileName="vTempFile" Sep="#9" Quote="#34">
    <!-- ... -->
    </OpenFile>
    <File Name="vTempFile" Do="Delete" />
  </IF>
</FOR>
```

2.7 FetchError / OnError

Mit FetchError/OnError kann eine Fehlerbehandlung etabliert werden. Ohne Fehlerbehandlung wird beim Auftreten eines Fehlers die Ausführung abgebrochen und damit auch allfällig begonnene Transaktionen.

Achtung: Wird FetchError / OnError verwendet, dann liegt eine explizite Transaktionssteuerung vor und alle Skriptelemente sollten in FetchError / OnError geklammert werden.

Wenn ein Fehler auftritt, dann wird die Ausführung nach OnError fortgesetzt. Wenn das OnError Element ohne Fehler erreicht wird, dann wird die Ausführung nach dem </FetchError> fortgesetzt. Fehlt OnError, dann wird die Ausführung abgebrochen.

```
<FetchError>
  <!-- Hier kommen Aktionen, welche unter Umständen einen Fehler auslösen können --
>
<OnError />
  <!-- Hier kommen wir im Fehlerfall -->
  <LogEntry Value="'Fehler aufgetreten: ' + varLastError "/>
</FetchError>
```

Mit der Fehlerbehandlung wird auch eine explizite Transaktionssteuerung eingeführt:

```
<FetchError BisUpdate="yes">
  <!-- Fehlerüberwachter Code -->
<OnError Transaction="abortAtBegin" />
  <!-- hier ist keine Transaktion mehr geöffnet -->
</FetchError>
```

Bedeutet, dass alle Elemente in FetchError in einer Schreibtransaktion ausgeführt werden. Im Falle eines **Deadlocks** der Datenbank wird die Transaktion abgebrochen und wiederholt (ab V4.9.8). Das bedeutet, dass **alle Befehle** zwischen <FetchError> und </FetchError> bzw. <OnError> wiederholt werden. Dies muss beim Entwickeln der Transaktion berücksichtigt werden (z.B. müssen Zähler nach <FetchError> initialisiert werden).

Die Anweisungen zwischen OnError und dem Ende des FetchError werden nur im Fehlerfall ausgeführt. Mit dem Attribut `Transaction` (ab V4.9.3) steuert man ob die Transaktion gleich beim Erreichen des OnError abgebrochen wird (abortAtBegin) oder ob die Transaktion für die Anweisungen nach dem OnError beibehalten werden soll und erst am Schluss abgebrochen (abortAtEnd) oder ausgeführt (commitAtEnd) werden soll.

Achtung: Transaktionen werden nur abgebrochen oder beendet, wenn in `FetchError` eine Transaktion begonnen wurde (`BisUpdate` gesetzt).

```
Transaction="abortAtBegin" //default
Transaction="abortAtEnd"
Transaction="commitAtEnd"
```

Vor V4.9.3 wird nur die obsolete Form `Abort="yes"` (=abortAtBegin) und `Abort="no"` (Transaktion wurde gar nicht geschlossen) unterstützt.

2.8 File

Mit File können Operationen auf Dateien ausgeführt werden. Es gibt dazu folgende Attribute:

Name:	Datei, bzw. Quelldateiname als Ausdruck.
Target:	Zieldatei als Ausdruck, dies wird zB. beim Kopieren benötigt.
Do	ein Wert von: Copy, Delete, Readonly, Readwrite (ab V4.5.1)

```
<Variables>
  <Var Name="fsource"  Value="='C:\Temp\Source.txt'"/>
  <Var Name="ftarget1" Value="='C:\Temp\Target1.txt'"/>
  <Var Name="ftarget2" Value="='C:\Temp\Target2.txt'"/>
</Variables>

<Script>
  <File Name="ftarget1" Do="Readwrite"/>
  <File Name="fsource" Target="ftarget1" Do="Copy"/>
  <File Name="ftarget1" Do="Readonly"/>

  <File Name="ftarget2" Do="Readwrite"/>
  <File Name="ftarget2" Do="Delete"/>
  <File Name="fsource" Target="ftarget2" Do="Copy"/>
</Script>
```

2.9 FileDialog

Ein FileDialog kennt folgende Attibute:

Open:	yes = Öffnen Dialog (default) no = Speichern Dialog
ResultVar	Variable in welche der Dateiname gespeichert wird. Wenn der Benutzer den Dialog abbricht, entspricht dies einem Leertext (""). Wenn mehrere Dateien ausgewählt werden, dann werden die Dateinamen mit Semikolon getrennt.
Titel	Titel des Dialoges
DirRegistry	RegistryKey wo das Verzeichnis abgespeichert, bzw gelesen wird.
Filter	Expression für die Definition der Dateiendungen
DefaultExt	Text für die Standard-Dateiendung

Mit folgenden Boolean-Attributen können die Dialogoptionen eingestellt werden:

ReadOnly	false	Ignoriert ab v5.5.6
OverwritePrompt	false	
HideReadOnly	true	Ignoriert ab v5.5.6
NoChangeDir	false	
ShowHelp	false	Ignoriert ab v5.5.6

NoValidate	false	
AllowMultiSelect	false	
ExtensionDifferent	false	
PathMustExist	false	
FileMustExist	false	
CreatePrompt	false	
ShareAware	false	
NoReadOnlyReturn	false	
NoTestFileCreate	false	
NoNetworkButton	false	Ignoriert ab v5.5.6
NoLongNames	false	Ignoriert ab v5.5.6
OldStyleDialog	false	Ignoriert ab v5.5.6
NoDereferenceLinks	false	
EnableIncludeNotify	false	Ignoriert ab v5.5.6
EnableSizing	true	Ignoriert ab v5.5.6
DontAddToRecent	false	
ForceShowHidden	false	
PickFolders	false	Erst ab v5.5.6

Minimal:

```
<FileDialog ResultVar="ipf"/>
```

Öffnen-Dialog:

```
<FileDialog Open="yes"
  ResultVar="ipf"
  Title="Mobiliar importieren"
  Filter="'Mobiliar (*.txt)|*.txt|Alles (*.*)|*.*'"
  DefaultExt="txt" />
```

Speichern-Dialog:

```
<FileDialog Open="no"
  ResultVar="ipf"
  Title="Mobiliar exportieren"
  Filter="'Mobiliar (*.txt)|*.txt|Alles (*.*)|*.*'"
  DefaultExt="txt"
  OverwritePrompt="true" />
```

Mehrfachauswahl:

```
<FileDialog Open="yes"
  ResultVar="vOrigFiles"
  Title="Dateien einlesen"
  Filter="'Excel-Dateien (*.xls*)|*.xls*|Alle Dateien (*.*)|*.*'"
  DefaultExt="xls*"
```

```
  AllowMultiSelect="true"/>
<IF Condition="STRLEN(vOrigFiles) &gt; 0">
  <FOR Variable="i" From="0" To="STRPOS(';', vOrigFiles, 0)">
    <LogEntry Type="information" Value="FIELD(vOrigFiles, i, ';')"/>
  </FOR>
</IF>
```

2.10 FOR… (ab v4.5)

<FOR> kann für Schleifen verwendet werden, bei welcher Anzahl Durchlauf schon am Anfang bekannt ist. Zudem kann damit implizite eine Schleifenvariable definiert werden.

```
<FOR Variable="i" From="0" To="adoFieldCount-1">
  <ADOgetField Id="ado" GetByNumber="i" Variable="n" Info="name" />
</FOR>
```

Default für From ist 0 und wenn die Schleifenvariable nicht benötigt wird, dann kann diese auch weggelassen werden. N.B. Die From/To Ausdrücke werden nur einmal, evaluiert.

2.11 ForEachFile

`<ForEachFile>` führt für jede Datei, welche dem Muster (FilePattern) entspricht alle enthaltenen Elemente aus. Zuvor wird der gesamte Dateiname der Variablen ToVar zugewiesen. Wenn man auch die Verzeichnisse innerhalb des Verzeichnisses erfassen möchte, dann muss im Attribut `Options "recursive"` gesetzt sein.

Attribute von ForEachFile:

`Dir`	Expression mit dem Verzeichnis.
`FilePattern`	Expression mit dem Dateinamen, welcher auch ‚*' und ‚?' enthalten darf.
`ToVar`	Name der Variable in welcher der vollständige Dateiname zurückgegeben wird. Die Anweisungen innerhalb von ForEachFile werden für jede Datei einmal ausgeführt und diese Variable ist entsprechend gesetzt
`Options`	kann folgende Flags enthalten: `recursive`: Es werden auch unterverzeichnisse untersucht. `dirOnly`: Es werden nur Verzeichnisse berüksichtigt (ohne ‚.', ‚..') `noDir`: Es werden nur Dateien berüksichtigt Wenn weder dirOnly, noch noDir gesetzt ist, dann werden Dateien und Verzeichnisse berücksichtigt, auch ‚.' und ‚..'
`Sort`	Steuert die Reihenfolge, in der die Namen zurück gegeben werden. Das Attribut hat einen der folgenden Werte: `none`: beliebiege Reihenfolge (default) `asc`: aufsteigend (a vor b) `desc`: absteigend (b vor a)

`dirOnly, noDir und Sort` gibt es ab V4.11.1

Beispiele:

```
<ForEachFile Dir="'K:\Plaene'"  FilePattern="'*.dwg'" ToVar="fpn">
  <LogEntry Value="EXTRACTNAME(fpn)"/>
</ForEachFile>
```

```
<ForEachFile Dir="'P:\Notizen'" Options="recursive, noDir"
          Sort="asc" FilePattern="'*.txt'" ToVar="fpn">
  <LogEntry Value="fpn"/>
</ForEachFile>
```

2.12 ForEachObject / BuildGroup

ForEachObject führt die enthaltenen Elemente für jedes Eingangsobjekt genau einmal aus.
Die Objektmenge am Ende von ForEachObject kann durch das Attribut `OutIs` gesteuert werden.

`OutIs = "in" | "evaluated" | "empty"`
Default: `in`

`in`:	Dieselbe Menge wie vor dem ForEachObject.
`evaluated`:	Die Vereinigungsmenge aller Objekte am Ende von ForEachObject.
`empty`:	Leere Menge

```
<ForEachObject OutIs="in">
  <SetVar BisAttribute="Doc_Path" Name="fpn"/>
  <LogEntry Value="'Plan Name: '+ fn"/>
</ForEachObject>
```

Das **erste** Element innerhalb von ForEachObject kann ein BuildGroup Element sein. Wenn es ein BuildGroup Element gibt, dann wird für jedes Objekt zuerst die Elemente unterhalb von BuildGroup ausgeführt und danach der Ausdruck in GroupBy ausgewertet. Objekte mit demselben Wert ergeben eine Gruppe. Die Elemente von ForEachObject werden dann für jede Gruppe einmal ausgeführt.

```
<Navigation>
  START INSTANCES Person
</Navigation>
<ForEachObject>
  <BuildGroup GroupBy="SUBSTR(nn,1,1)">
    <SetVar Name="nn" BisAttribute="bisB_Lastname" />
  </BuildGroup>
  <SetVar Name="anz" BisAttribute="COUNT" />
  <LogEntry Value="'------ ' + FORMAT(anz, '%d')" />
  <ForEachObject>
    <SetVar Name="disp" BisAttribute="Object_Display_Kontext" />
    <LogEntry Value="disp" />
  </ForEachObject>
</ForEachObject>
```

2.13 ForEachProperty (ab v4.4)

ForEachProperty iteriert alle existierenden oder erlaubten Properties (ab Version 4.5.5 auch Assoziationen) des aktuellen Objekts. Der Name der Property wird in die Variable `NameToVar` und der Wert in der Variable `ValueToVar` gespeichert. Mit `Pattern` kann ein Ausdruck mit einem Namensmuster definiert werden (default = '*').

Ab Version 4.5.5 kann mit `PropertyTypes` bestimmt werden, ob Attribute ("`attributes`"), Assoziationen ("`associations`") oder beides ("`both`") ausgegeben werden soll (default = "`attributes`") Die Variable `ValueToVar` enthält bei Attributen den Attribut-Wert und bei Assoziationen die verknüpften Objekte.

Änderungen in Version 4.10.6

Mit `AllProperties` wird bestimmt, ob alle erlaubten (="yes") oder alle existierenden Eigenschaften (="no") ausgegeben werden (default = "no").

Mit `CaptionToVar` kann die externe Bezeichnung der Property in eine Variable abgelegt werden.

Die Angabe von `NameToVar`, `ValueToVar` und `CaptionToVar` ist optional.

```
<ForEachObject>
  <ForEachProperty Pattern="'MV_*'"
                   PropertyTypes="attributes"
                   NameToVar="vname"
                   ValueToVar="vvalue"
                   AllProperties="no">
    <XMLelement Name="vname">
      <XMLtext NilFlag="yes" Value="vvalue" />
    </XMLelement>
  </ForEachProperty>
</ForEachObject>
```

2.14 FTP connection get put (ab v4.7)

Connect erstellt eine FTP-Verbindung. Innerhalb dieser Verbindung können mit FTPput und FTPget Dateien transferiert werden. FTPput und –get beziehen sich über den Id Mechanismus mit der FTPconnection.

```
<FTPconnection Id="ftp" Host="'byrhst'"
               Username="'ada'" Password="'lovelace'">
    <FTPget Id="ftp" Source="'Hallo.txt'"  Target="'C:\Temp\Hi.txt'" />
    <FTPput Id="ftp" Source="'C:\Temp\9.wma'" Target="'music/B9.wma'" />
<FTPconnection >
```

2.14.1 FTPconnection

Id :	text
Host :	expression
Username :	expression
Password :	expression
Port :	expression integer default 21
Source..:	expression Quelldateiname
Target..:	expression Zieldateiname
CanOverwrite :	boolean (default = „no“)
Append :	boolean (default = „no“)

2.14.2 FTPget (download)

Source..:	expression Quelldateiname (remote)

Target..:	expression Zieldateiname (lokal)
CanOverwrite :	boolean (default = „no“)

2.14.3 FTPput (upload)

Source..:	expression Quelldateiname (lokal)
Target..:	expression Zieldateiname (remote)
Append :	boolean (default = „no“)

2.15 GetVar

Mit GetVar können Objektmengen, welche mit SetVar gespeichert wurden wieder abgerufen werden. Das Bedeutet die Eingangsmenge wird verworfen und die Ausgangsmenge entspricht der Menge von SetVar.

```
<GetVar Name='memory1'>
```

Beispiel:

```
<Navigation>
  START INSTANCES Doc_File
</Navigation>
<SetVar Name="allDocs"/>
<Navigation>
  CLASS Space
</Navigation>
<SetVar Name="i" Expression="0"/>
<ForEachObject>
  <SetVar Expression="i+1" Name="i"/>
</ForEachObject>
<LogEntry Value="'Jetzt '+FORMAT(i,'%d')+' (kein) Datei-Dokument'"/>
<GetVar Name="allDocs"/>
<SetVar Name="i" Expression="0"/>
<ForEachObject>
  <SetVar Expression="i+1" Name="i"/>
</ForEachObject>
<LogEntry Value="'Anazahl Datei-Dokumente : ' + FORMAT(i, '%d')"/>
```

2.16 IF ELSE

Ein IF besitzt eine Condition und kann ELSE Elemente enthalten. Ein ELSE besitzt auch eine Condition (=ELSIF) ausser beim letzten ELSE kann die Condition weggelassen werden

Beispiel:

```
  <IF Condition="plt = 30">
    <LogEntry Value="'Ein 30iger'"/>
  <ELSE Condition="plt = NULL"/>
    <LogEntry Value="'undefiniert !!!'"/>
  <ELSE Condition="plt = 0"/>
    <LogEntry Value="'Architektur (0er)'"/>
```

```
<ELSE/>
  <LogEntry Value="'typ code = ' + FORMAT(plt, '%d')"/>
</IF>
```

2.17 LogEntry

Mit LogEntry wird ein Log-Eintrag geschrieben. Die Log-Einträge werden in BisXds.exe angezeigt oder können mit `<Logfile Name="C:\Temp\log.txt"/>` (vgl. XDS_IntroductionReference.pdf) in eine Datei geschrieben werden.

2.17.1 Mit Type

Die Art des Eintrages kann durch Type gesteuert werden.

`Type = "error" | "warning" | "information" | "debug"`
Default: `"information"`

Der Wert wird als Expression in Value angegeben.

```
<LogEntry Type="warning" Value="'Wert: ' + IFNULL(nm, '-')"/>
```

In diesem Fall muss die Variable nm allenfalls einen Wert vom Typ Text haben.

Ab Version 5.1.1 Unterstützung von Exit-Code

Mit `Type="error"` wird eine Exception ausgelöst. Diese kann mit OnError abgefangen werden. Wenn zudem Value ein Integer ist, dann wird dieser Wert als Exit-Code gesetzt:

```
<LogEntry Type="error" Value="2" />
```

Kann in einer Batchdatei mit errorlevel ausgelesen werden:

```
echo Exit Code is %errorlevel%
```

2.17.2 Mit Options

Ab Version 4.10.6 gibt es ein Attribut `Options`. In diesem Fall wird der Wert immer ausgegeben, unabhängig des Debug-Levels. Jeder Eintrag impliziert den Eintrag always. Es können auch beliebige Kombinationen von Werten angegeben werden. Die Werte sind:

`always`	Wenn keine der anderen Optionen angegeben wird.
`header`	Der Eintrag erfolgt nur am Anfang der Logdatei.
`version`	Dem Eintrag wird die Versionsnummer beigefügt.
`process`	Dem Eintrag wird die Prozessnummer beigefügt.
`debug`	Dem Eintrag wird der Name der Debug-Umgebungsvariable und deren Wert beigefügt.
`logDir`	Dem Eintrag wird der Name der LogDir-Umgebungsvariable und deren Wert beigefügt.

Beispiel:

```
<Script>
  <LogEntry Options="header, debug" Value="'Das ist ein Titel'" />
  <LogEntry Type="information" Value="'Das sieht man mit Level Info'" />
  <LogEntry Options="always" Value="'Das sieht man immer'" />
```

```
</Script>
```

2.18 Logfile (ab v4.10.6)

Das Element innerhalb von Script ist nicht zu verwechseln mit dem Element ausserhalb. Letzteres entspricht der Doku: XDS_IntroductionReference.pdf und hat in diesem Fall nur ein Attribut: `Name`
`<Logfile Name="C:\Temp\log.txt"/>`

Dies entspricht innerhalb von `<Script>` dem Aufruf:

```
<Logfile  DirectoryName="'C:\Temp'"
          FileName="'log'"
          ExtensionName="'txt'"/>
```

Man beachte, dass alle Werte (ausser Level) Ausdrücke (Expressions) sind und Texte deshalb in einfachen Anführungszeichen geschrieben werden müssen. Logfile kennt folgende Attribute, welche alle fakultativ sind.

`Application` : Text

Name der Anwendung

`FileName` : Text

Logdateiname, bzw. das Muster für den Dateinamen. Folgende Werte werden ersetzt:
#NOW# das aktuelle Datum gemäss der Formatangabe in DateFormat
#ProcessID# die Prozessnummer
#Application# Name der Application
#IncrementingNumber# Laufende Nummer

`DirectoryName` : Text

Verzeichnis der Logdatei. Dieses wird normalerweise mit der Umgebungsvariablen **`BISXDS_LOGDIR`** gesetzt und kann mit diesem Attribut überschrieben werden.

`ExtensionName` : Text

Bezeichnung der Dateierweiterung.

`DateFormat` : Text

Wenn im Filename #NOW# verwendet wird, dann kann damit die Datum- Zeitangabe gesteuert werden. yyyy Jahr 4-stellig, mm Monat 2-stellig, dd Tag 2-stellig, hh Stunde 2-stellig, nn Minute 2-stellig.

`Move` : Boolean *)

Bestimmt, ob der Log-Dateiinhalt verschoben werden soll, wenn der Log-Dateiname wechselt.

`Append` : Boolean *)

Bestimmt, ob eine existierende Datei ersetzt wird, oder ob die neuen Einträge angehängt werden.

*) Es handelt sich hier auch um Ausdrücke. Die Konstanten für BOOLEAN sind TRUE bzw. FALSE, alles in Grossbuchstaben!

`Level` : error, warning, information oder debug ab V4.10.7

Setzt das Fehlerlevel. Dieses wird normalerweise mit der Umgebungsvariablen **`BISXDS_DEBUG_LEVEL`** gesetzt und kann mit diesem Attribut überschrieben werden.
Beispiel: `Level="information"`

`DeletePreLogFileLevel` : error, warning, information oder debug ab V4.11.2

Definiert den Debug-Level, ab welchem die alte LOG-Datei nicht gelöscht werden soll. (Default = warning)
Beispiel: `DeletePreLogFileLevel="information"`

Ein Beispiel in dem alle Attribute angegeben werden:

```
<Logfile
   Application           = "'XDS-BIS'"
   Move                  = "FALSE"
   Append                = "TRUE"
   FileName              = "'XDS_#NOW#'"
   DateFormat            = "'yyyy.mm'"
   DirectoryName         = "'%TEMP%\BIS'"
   ExtensionName         = "'log'"
   Level                 = "information"
   DeletePreLogFileLevel = "information" />
```

2.19 MailSend / Attachment / Text

Mit MailSend kann eine Mail versendet werden.

Attribute von MailSend:

Host : Text/=Exp **) Mail Server

ContentType :Text/=Exp**) Content-Type der Mail, zB. text/plain oder text/html 1)

From : Expression Absender

To : Expression Empfänger

CC : Expression CC Empfänger

BCC : Expression BCC Empfänger

ReplyTo Expression ReplyTo Empfänger

Subject : Expression Betreff

Contents : Expression Der gesamte Inhalt der Mail.

Port : Expression Portnummer (Default = 25)

User : Text Anmeldung (Login) am Mail Server (optional) ab V4.5.4

User : Expression Anmeldung (Login) am Mail Server (optional) ab 5.7.3

PWD : Text Password wenn User angegeben wurde ab V4.5.4

PWD : Expression Password wenn User angegeben wurde ab V5.7.3

AuthType : 'None', 'Default', 'SASL' *)
ohne User wird als default ‚None', sonst ‚Default' verwendet.

useTLS : NoTLSSupport', 'UseImplicitTLS', 'UseRequireTLS', 'UseExplicitTLS' *)

*) ab V4.9.0 die verwendeten Werte werden im Debug-Modus angezeigt. Für mehr Info zur Bedeutung kann die Attributbezeichnung und indy delphi verwendet werden, also google nach `useTLS indy delphi`

**) ab V4.10.0 Text/=Exp bedeuted, dass mit einem ‚=' Zeichen an erster Stelle eine Expression geschrieben werden kann. Der Host im nachfolgenden Beispiel llesse sich auch so schreiben:
Host="= 'mail.byron.ch' "

Beispiel:

```
<MailSend  Host="mail.byron.ch"
           ContentType="text/plain; charset=iso-8859-1"
           From="'support@byron.ch'"
           To="'info@byron.ch'"
           Subject="'meine erste mail'"
           Contents="txtBody">
```

Dabei ist `txtBody` eine Variable, welche den gesamten Mail Text enthält.

Im **ContentType** wird auch das Encoding des Inhalts angegeben, z.B. "text/html; charset=iso-8859-1" wenn im ContentType ‚text' vorkommt und kein charset, dann wird charset=us_ascii angenommen!, dies kann dann der Grund für fehlende Umlaute sein. Vgl. hierzu TransformXSL

Attachment / Text ab V4.5.4

MailSend kann Attachment Elemente und/oder ein Text Element enthalten. Ab Version 4.8.3 kann MailSend auch alle anderen Elemente enthalten, somit können auch dynamisch viele Attachments erzeugt werden (z.B. Attachment innerhalb eines ForEachObject). Ein solcher Anhang wird definiert durch eine Datei (der Anhang selbst), Ort des Anhangs und den Typ. Der Text ist kein Anhang, sondern der Mailinhalt, falls die Mail einen Anhang besitzt.
Siehe auch Beispiel Html E-Mail versenden.

Attribute von Attachment / Text

Name : Expression Dateiname

ContentType: Text/=Exp. Content-Type der Mail, zB. image/jpeg 1)

ContentTransferEncoding Content-Transfer-Encoding zB. 8bit 1)

Attribute nur von Attachment

ContentDisposition: Text /=Exp “inline” oder “attachment” 2)

Attribute nur von Text

Contents: Expression Der Ausdruck enthält den Mailtext

Ein Text Element ist ein ‚Attachment' inline, welches den Inhalt aus ‚Contents' anstatt der Datei (Name) bezieht.

Beispiel:

```
<Attachment Name="fn"
            ContentDisposition="inline"
            ContentType="text/html; charset=iso-8859-1"/>
```

Dabei muss es eine Variable fn geben, welche auf eine existierende Datei verweist.

In MailSend kann der `Contents` weggelassen werden und dafür ein Attachment mit ContentDisposition="inline" definiert werden. Mit dieser Technik kann man den Mail Inhalt in einer Datei, statt in einer Variablen zusammenstellen.

Wenn ein Attachment definiert ist, wird im MailSend für ContentType 'multipart/mixed' eingesetzt.

1. http://de.wikipedia.org/wiki/Content-Type
   http://de.selfhtml.org/diverses/mimetypen.htm
   http://en.wikipedia.org/wiki/MIME
2. Internet Suche nach Content-Disposition

Muster einer Mail mit PDF und Fotoanhang:

```
<MailSend ContentType="multipart/mixed">
  <Text Contents="vContents"
        ContentDisposition="inline"
        ContentType="text/html; charset=iso-8859-1"/>
  <Attachment Name="fnj"
        ContentDisposition="attachment"
        ContentType="image/jpeg"/>
  <Attachment Name="fnp"
        ContentDisposition="attachment"
        ContentType="application/pdf"/>
<MailSend/>
```

Siehe auch Beispiel Html E-Mail versenden.

Wichtiger Hinweis für das Versenden von Html E-Mails:

Die meisten SMTP Mailserver setzen eine *Maximum Body Line Length* von 1000 Zeichen durch und verkürzen die Zeilen durch Einfügen von CR-LF in den Text. Aus diesem Grund sollte, um ungültigen Html-Text zu vermeiden, entweder sichergestellt werden, dass keine Zeile zu lang wird, oder dass eine „multipart/alternative-E-Mail" erstellt wird.

Bsp. für eine „multipart/alternative-E-Mail":

```
<MailSend Host="mail.byron.ch"
          ContentType="multipart/alternative; charset=iso-8859-1"
          From="sender" To="receiver" CC="cc"
          Subject="subject" >
  <Text Contents="subject + ', Inhalt siehe HTML-Text'"
        ContentType="text/plain; charset=iso-8859-1"/>
  <Text Contents="mailTextHtml"
        ContentType="text/html; charset=iso-8859-1"/>
</MailSend>
```

2.20 MenuAction

Führt eine MenuAction in BIS aus. Eine MenuAction kann zum Beispiel das Ausführen eines Reports sein. Die Input Objekte werden der MenuAction übergeben. Die MenuAction wird über den Namen identifiziert (der Name ist ein Text, keine Expression). Für Parameter gibt es drei Mechanismen:

1. Parameter im para=Wert Stil: der MenuAction werden als Attribute spezifiziert. Es werden also alle XLM-Attribute ausser Name als Parameter übergeben.

```
<MenuAction Name="GraphicReportDIN277" Printer="?"/>
```
Hier wird ein Parameter `Printer=?` erzeugt.

Wenn in Parameter Variablen (Namen) vorkommen, werden diese durch den Wert ersetzt.

2. (ab Version 4.8.5) Parameter in XML Notation: Die enthaltene XML Struktur wird als Parameter übergeben. Der Text eines XML Elementes wird als Expression evaluiert, wenn der Text mit einem = beginnt.

```
<MenuAction Name="TextReport">
  <Output>
    <File Type="pdf">=OSENV("Temp") + 'mytxt.pdf'</File>
  </Output>
</MenuAction>
```

3. (ab Version 4.8.6) Wenn das Element Text enthält (aber keine XML-Elemente), dann wird dieser Text als Parameter verwendet. Wenn der Text mit einem = beginnt, dann wird der übrige Text als Expression evaluiert.

```
<MenuAction Name="ShowView">=fn + '.pdf'</MenuAction>
```

Parameter ist Expression Wert von fn, mit Endung pdf.

2.21 Navigation

Die Eingangsobjekte werden als Menge auf die (Filter) Navigation angewendet. Dadurch ergibt sich die Ausgangsmenge. Möchte man die Filternavigation auf jedes Objekt einzeln anwenden, dann müsste man die Navigation innerhalb eines <ForEachObject> definieren.

Beispiel:

```
<Navigation>
  VIA Object_To_Children
  CLASS Room
</Navigation>
```

2.22 OpenFile / Next

Mit OpenFile kann eine Textdatei zum Lesen geöffnet werden, oder einen neue Datei angelegt werden. Es können auch UTF-8 Dateien gelesen werden (ab V4.8.4) . Wenn man eine UTF-8 Datei schreiben möchte, dann setzt man das Attribut `encoding="UTF-8"` im Element OpenFile.

```
<OpenFile Mode="read" FileName="fn" RecVar="rec">
```

`Id`	Text Identifier für die Datei. Mit diesem Identifier können sich andere Operationen, zB. Next auf diese Datei beziehen. Default Wert ist "".
`Mode`:	`"read" | "create" | "append"`
`FileName`:	Variable, welche den Dateinamen enthält
`RecVar`:	Variable, in welche der Record (Zeile) geschrieben wird. Wenn `RecVar` weggelassen wird, dann muss der Record explizit mit `Next` gelesen werden.
`Sep`	Definiert einen Separator dies kann beim lesen in Zusammenhang mit `Quote` nützlich sein. Dadurch ist es möglich Felder mit Zeilenumbrüchen zu lesen. `Sep` hat **keinen** Vorgabewert. **N.B.**: Die Angabe von `Sep` bei `OpenFile` ist nur sinnvoll zusammen mit `Quote`.
`Quote`	Definiert das Anführungszeichen, welches verwendet wird wenn Zeilenumbrüche innerhalb eines Feldes vorkommen. Ebenso werden diese Zeichen entfernt, wenn das Feld damit beginnt und endet. Quote hat **keinen** Vorga-

bewert.
N.B.: `Quote` **nie ohne** `Sep` verwenden. `Quote` wird ignoriert, wenn `Sep` nicht explizit gesetzt wird.
Vgl. Beispiel.

`encoding` ab V4.10.0 nur für Mode create mögliche Werte: `UTF-7`, `UTF-8`, `UTF-16`, `ISO-8859-1`
N.B. ab Version 4.8.4 waren nur die Werte `ISO-8859-1` oder `UTF-8` zulässig

Beispiel Definition der Datei:

```
<Variables>
  <Var Name="fn" Value="=OSENV('temp') + '\aa.txt'" />
</Variables>
```

Beispiel: Lesen einer Textdatei mit RecVar. Die Elemente in OpenFile werden für jeden Record einmal ausgeführt, dabei wird zuerst der Variable den Wert des Records (Zeile) zugewiesen.

```
<Script>
  <LogEntry Value="'lese: ' + fn"/>
  <OpenFile Mode="read" FileName="fn" RecVar="rec">
    <LogEntry Value="'Record ' + rec"/>
  </OpenFile>
  <LogEntry Value="'fertig: ' + fn"/>
</Script>
```

Beispiel: Lesen einer Textdatei welche aus Excel kommt und mehrzeilige Felder enthalten kann. Sep und Quote werden normalerweise immer zusammen definiert.

```
<Script>
  <File Do="delete" Name="ftemp" />
  <ExcelToText Excel="ipf" Text="ftemp" />
  <OpenFile Mode="read" FileName=" ftemp " RecVar="rec"
                 Sep="#9" Quote="#34" >
  </OpenFile>
</Script>
```

Beispiel: Lesen einer Textdatei durch explizites Abfragen des Records. Die Verknüpfung von OpenFile mit Next wird durch eine ‚Id' hergestellt. Damit kann OpenFile auch verschachtelt sein. Wenn die Id weggelassen wird, dann ist diese leer.

```
<Script>
  <LogEntry Value="'lese: ' + fn"/>
  <OpenFile Mode="read" FileName="fn">
    <Next Variable="rec"/>
    <WHILE Condition="NOT (rec = NULL)">
       <LogEntry Value="'Record ' + rec"/>
       <Next Variable="rec"/>
    </WHILE>
  </OpenFile>
  <LogEntry Value="'fertig: ' + fn"/>
</Script>
```

Beispiel: Lesen und Schreiben einer Textdatei.

```
<Script>
  <LogEntry Value="'lese: ' + fi"/>
```

```
  <LogEntry Value="'schreibe: ' + fo"/>
  <SetVar Name="i" Expression="0"/>
  <OpenFile Id="out" Mode="create" FileName="fo" RecVar="rec">
    <OpenFile Id="in" Mode="read" FileName="fi">
      <Next Id="in" Variable="rec"/>
      <SetVar Name="i" Expression="i+1"/>
      <SetVar Name="recOut" Expression="FORMAT(i, 'rec %d) ') + rec"/>
      <Next Id="out" Variable="recOut"/>
      <LogEntry Value="'Record ' + recOut"/>
    </OpenFile>
  </OpenFile>
  <LogEntry Value="'fertig: ' + fo"/>
</Script>
```

Durch `RecVar` beim `OpenFile`-Lesen wird der Inhalt dieses Elementes für jede Zeile aufgerufen. Wenn RecVar im OpenFile weggelassen wird, dann entfällt diese implizite Schleife und die Zeilen müssen expizit gelesen werden:

```
<Script>
  <LogEntry Value="'lese: ' + fi"/>
  <LogEntry Value="'schreibe: ' + fo"/>
  <SetVar Name="i" Expression="0"/>
  <OpenFile Id="in" Mode="read" FileName="fi">
    <OpenFile Id="out" Mode="create" FileName="fo">
      <Next Id="in" Variable="rec"/>
      <WHILE Condition="rec &lt;&gt; NULL" >
        <SetVar Name="i" Expression="i+1"/>
        <SetVar Name="recOut" Expression="FORMAT(i, 'rec %d) ') + rec"/>
        <Next Id="out" Variable="recOut"/>
        <LogEntry Value="'Record ' + recOut"/>
        <Next Id="in" Variable="rec"/>
      </WHILE>
    </OpenFile>
  </OpenFile>
  <LogEntry Value="'fertig: ' + fo"/>
</Script>
```

Beim Schreiben ist zu beachten, dass ein NULL Wert im Next zum Beenden des gesamten <OpenFile> führt.

2.23 Progress

Mit Progress kann eine Fortschrittsanzeige zur Information des Benutzers dargestellt werden.

Attribute:
Titel : Expression Text
TextOben : Expression Text
TextUnten : Expression Text
Position : Expression Float 0.0 – 1.0 definiert die Position des Fortschrittsbalken
Refresh : yes / no (Default yes) mit yes wird noch ein 'ProcessMessages' durchgeführt, damit werden die Chancen, für ein erneutes Zeichnen erhöht.

Beispiel

```
  <Progress Title="'Jetzt geht es schnell'"
            TextOben="'Geschwindigkeit'"
```

```
          Position="0.05"/>
  <SetVar Name="j" Expression="0"/>
  <WHILE Condition="j &lt; 50000">
    <Progress Position="0.05 + 0.94 * (j / 50000)"/>
    <SetVar Name="j" Expression="j + 1"/>
  </WHILE>
```

2.24 ReadXML / ModifyXML / SetSelectionNamespace / ForEachElement / NodeToVar / TransformXSL

Durch das ReadXML Element wird eine XML Datei gelesen, welche innerhalb des Elementes gelesen werden kann. Die XML Daten könnnen alternativ auch mit einer URL über das Internet bezogen werden. Das ForEachElement definiert eine Liste von XML-nodes (Knoten) durch einen XML-Path (SelectNode). Für jeden Knoten, welche dem SelectNode entspricht, werden die Elemente von ForEachElement ausgeführt.
Innerhalb des ForEachElement gibt es ein aktuelles Element (node), welches mit NodeToVar gelesen werden kann.

Ab Version 4.4.6 kann statt ReadXML auch ModifyXML verwendet werden, dann wird die Datei am Schluss wieder gespeichert.

```
<ReadXML FileName="variable">…</ReadXML>
<ForEachElement SelectNode="x-path expression">..</ForEachElement>
<NodeToVar VarName="variable" NodeAttribute="XML-Attribut Name"/>
```

Durch `<Navigation>` Elemente können Daten in ByronBIS importiert werden (modifizierende Filternavigation).

Um Werte im XML zu ändern können `<XMLelement>` `<XMLtext>` `<XMLattribute>` Elemente verwendet werden vgl. Kapitel WriteXML. Dies wird i.A. für `<ModifyXML>` gebraucht.

2.24.1 TransformXSL (ab v4.5.1)

Mit `<TransformXSL>` kann der aktuelle Knoten auf ein Stylesheet angewendet werden. Das Stylesheet befindet sich innerhalb des `<TransformXSL>` Knotens. Der resultierende Text kann in eine Datei oder in eine Variable gespeichert werden.

Attribute von TransformXSL:

Id	Default "" Damit definiert man auf welche Datei sich der Befehl bezieht
ToVar	text Variablenname
Output	expression Die ausgewertete Expression ist der Output Dateiname
Flag	text. Wenn der Wert auf ‚transformNode' gesetzt ist, wird auch diese Routine verwendet (bzw. transformNodeToObject). Mit diesen Methoden gibt es Schwierigkeiten mit match="/". Wenn Flag weggelassen wird, dann wird aus dem node ein Document gemacht und darauf Transform angewendet. Vor Version 4.8.0 gibt es kein Flag, aber die Ausführung entspricht `Flag="transformNode"` Wenn Flag nicht gesetzt ist, dann beachte man die korrekte Steuerung für das encoding des outputs. http://www.w3schools.com/xsl/el_output.asp

Beispiele gibt es am Schluss: HTML Mail senden / Termin-Transformieren

```
<TransformXSL ToVar="myHtml">
  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
    <xsl:output method="html" encoding="iso-8859-1" indent="yes"/>
    <xsl:template match="Person">
       <html> … </html>
    </xsl:template>
  </xsl:stylesheet>
</TransformXSL>
```

N.B. xsl:output ist notwendig wegen der Zeichencodierung des Resultats. Wird kein Encoding angegeben, dann verwendet die Transformation UTF-16, welches in Mailsend zu **Fehlern** führt.
-> als Encoding „utf-8“ oder besser „iso-8859-1“ verwenden.

N.B. Wird TransformXSL zum Senden einer HTML-Mail verwendet, dann muss das in xsl:output angegebene Encoding mit dem Encoding des Mailinhalts (ContentType) übereinstimmen.

Wenn man den Text nicht in die Variable ‚myHTML' gespeichert haben möchte, sondern in eine Datei, so kann der Dateiname als Expression angegeben werden.

```
<TransformXSL Output="fn">
  <xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" encoding="iso-8859-1" indent="yes"/>
  </xsl:stylesheet>
</TransformXSL>
```

Da Output eine Expression enthält, ist hier fn eine Variable, zB:

```
<Var Name="fn" Value="=OSENV('Temp') + '\_auftraege.html'"/>
```

Beispiel XML Datei einlesen (generiert durch WriteXML)

```
<Variables>
    <Var Name="fx" Value="=OSENV('temp') + '\aa.xml'" />
</Variables>
<Script>
  <SetVar Name="i" Expression="0"/>
  <ReadXML FileName="fx">
    <ForEachElement SelectNode="'//ab'">
      <SetVar Name="i" Expression="i+1"/>
      <NodeToVar VarName="v1" NodeAttribute="ID"/>
      <LogEntry Value="'gelesen ID: ' + v1"/>
    </ForEachElement>
  </ReadXML>
  <LogEntry Value="'xml gelesen: '+fx+FORMAT(i,' Anzahl nodes: %d')"/>
</Script>
```

Beispiel XML Datei ändern (ModifyXML ab V 4.4.6)

```
<Variables>
    <Var Name="fx" Value="=OSENV('temp') + '\aa.xml'" />
</Variables>
<Script>
  <SetVar Name="vID" Expression="3"/>
  <ModifyXML Id="file" FileName="fx">
     <ForEachElement Id="file"
              SelectNode="'//ab[@ID='+ FORMAT(vID,'%d') + ']'">
       <XMLattribute Id="file" Name="'name'" Value="'MEIER'"/>
```

```
      <XMLtext Id="file"
               Value="FORMAT(vID,'%d') + ' geändert: MEIER'"/>
      <LogEntry Value="'Knoten geändert'"/>
    </ForEachElement>
  </ModifyXML>
</Script>
```

2.24.2 ReadXML / ModifyXML

Id	Default "" Diese Id muss von den Operatoionen verwendet werden
RootName	text: Root Elementname des Dokuments
FormattedOutput	boolean: default "no" macht nur bei ModifyXML Sinn (ab v4.4.6)
FileName	expression: Dateiname alternativ zu URL
URL	expression: URL alternativ zu FileName

SetSelectionNamespace (ab v4.9.2)

Mit SetSelectionNamespace können die verwendeten Namespaces angegeben werden.
Dies ist für XPatch-Ausdrücke notwendig, welche Namespaces beinhalten.
Das Element ist nur für ReadXML und ModifyXML verfügbar.

Name	expression: Namespace z.B. "xmlns:xs"
URI	expression: URI des Namespace z.B. "http://www.w3.org/2001/XMLSchema"

Beispiel:

```
  <ReadXML FileName="vXMLFileName">
    <SetSelectionNamespace Name="'xmlns:xs'"
                           URI="'http://www.w3.org/2001/XMLSchema'"/>
    <ForEachElement SelectNode="'//xs:string'">
      ...
    </ForEachElement>
  </ReadXML>
```

ForEachElement

Id	Default "" Damit definiert man auf welche Datei sich der Befehl bezieht
SelectNode	expression: XML path

NodeToVar

Id	Default "" Damit definiert man auf welche Datei sich der Befehl bezieht
SelectNode	expression: Damit kann ausgehend vom aktuellen Element ein anderes Element spezifiziert werden. Wenn SelectNode fehlt, wird das aktuelle Element gelesen.
AcceptNoNode	boolean: default="no" Diese Einstellung hat nur eine Bedeutung, wenn SelectNode verwendet wird. Wenn damit kein Element gefunden wird, ergibt dies einen Fehler, ausser wenn AcceptNoNode="yes" spezifiziert wird. Die Variable wird auf NULL gesetzt. (vor V 4.4.5 blieb die Variable unverändert). Ab V 4.10.6 kann damit auch ein Fehler für mehrere Nodes vermieden werden.

EmptyValue	expression: Wenn der Wert leer oder NULL ist, kann hier ein Wert definiert werden, der dann verwendet wird. Dadurch können Default-Werte einfacher gesetzt werden. (ab v4.4.6)
VarName	string: Name der Variable in welcher der Wert abgelegt wird
NodeAttribute	string: Name des XML Attributes, welches gelesen wird. Besonderheiten: - wenn NodeAttribute fehlt wird der Text des XML-Elements gelesen. - wenn NodeAttribute = "-", dann wird der Elementname gelesen (>= v4.4.5). - wenn NodeAttribute mit einem "=" beginnt, dann wird der restliche Text als Expression interpretiert (>= v5.5.10)

Beispiele:

```
<ForEachElement SelectNode="'/rooms/room'">
  <NodeToVar VarName="vId" NodeAttribute="ID" />
  <ForEachElement SelectNode="'owner'">
    <NodeToVar VarName="vOwn" />
  </ForEachElement>
</ForEachElement>
```

Wenn es genau einen owner in einem room gibt:

```
<ForEachElement SelectNode="'/rooms/room'">
  <NodeToVar VarName="vId" NodeAttribute="ID" />
  <NodeToVar VarName="vOwn" NodeAttribute="ID" SelectNode="'owner'" />
</ForEachElement>
```

Wenn es höchsten einen owner in einem room gibt:

```
<ForEachElement SelectNode="'/rooms/room'">
  <NodeToVar VarName="vId" NodeAttribute="ID" />
  <NodeToVar VarName="vOwn" NodeAttribute="ID"
             SelectNode="'owner'" AcceptNoNode="yes" />
  <IF Condition="vOwn = NULL">
    <LogEntry Type="info" Value="'keinowner für ' + vId" />
  <ELSE/>
    <LogEntry Type="info" Value="vId + ' gehört ' + vOwn" />
  </IF>
</ForEachElement>
```

Alle Elemente unterhalb eines Raums lesen und entsprechend den Elementnamen Objekte erzeugen. Elementnamen werden durch NodeAttribute="-" gelesen.

```
<ForEachElement SelectNode="'/rooms/room
  <NodeToVar VarName="vId" NodeAttribute="ID" />
  <ForEachElement SelectNode="'*'">
    <NodeToVar VarName="vSub" NodeAttribute="-" />
    <SetVar Name="vSubClass" Expression="'PREFIX_' + vSub " />
    <Navigation>
      CREATE 1 DATAOF vSubClass (START VALUE bisB_SpaceID DATAOF vId)
    </Navigation>
  </ForEachElement>
```

```
</ForEachElement>
```

Mit folgendem XML wird ein ‚PREFIX_Drucker' und ein ‚PREFIX_Wandanstrich Objekt unterhalb des Raumes 0815 angelegt:

```
<rooms>
  <room ID="0815">
    <Drucker />
    <Wandanstrich />
  </room>
</rooms>
```

2.25 ReceiveGlobalNotifyCode / ReceiveAllGlobalNotify… (ab v4.8)

Mit ReceiveGlobalNotify können GlobalNotify Aufrufe (vgl. Methodensprache) abgefangen werden.

Code:	integer; Enthält den opCode der Notifikationen die man empfangen möchte (vgl. opCode in Methodensprache GlobalNotify). Bei negativen Werten werden alle opCodes berücksichtigt. Default = -1¨
TimeOut:	integer; maximale Wartezeit in Millisekunden. Negative Werte bedeuten unendlich langes Warten. Dies ist ein MUST-Attribut, es gibt also keine Default.
CodeVar	Name der Variable, in welche der Wert des Code geschrieben wird.

Für ReceivGlobalNotify gibt es 3 Anwendungvarianten:

1. Nur einen bestimmten Code abwarten:
   `<ReceiveGlobalNotifyCode Code="2" TimeOut="5000"/>`
   jetzt sind die Objekte gesetzt.
2. Alle GlobalNotify abfangen und dann je nach Code reagieren:
   `<ReceiveAllGlobalNotify TimeOut="8000" CodeVar="cd">`
   jetzt ist die Variable cd mit dem Code (integer) gesetzt und
   alle Objekte mit dem Code cd.
   `</ReceiveAllGlobalNotify>`
   Der Inhalt von `ReceiveAllGlobalNotify` wird für jeden auftretenden opCode einmal mit den entsprechenden Objekte durchlaufen. Nach dem Element wird die Objektmenge zu allen vorgekommenen Objekten gesetzt.
3. Alle GlobalNotify abfangen und unabhägig vom Code etwas machen
   `<ReceiveAllGlobalNotify TimeOut="8000" />`
   jetzt sind die Objekte gesetzt, auf den Code hat man keinen zugriff

Beispiel zu 1

```
<ReceiveGlobalNotifyCode Code="2" TimeOut="5000"/>
<SetVar Name="cnt" BisAttribute="COUNT"/>
<ForEachObject>
  <SetVar Name="v" BisAttribute="Object_Display_Kontext"/>
  <LogEntry Value="'Objekt: ' + v"/>
</ForEachObject>
```

Beispiel zu 2

```
<ReceiveAllGlobalNotify TimeOut="8000" CodeVar="cd">
  <SetVar Name="cnt" BisAttribute="COUNT"/>
  <LogEntry Value="FORMAT(cd, 'Code: %d') + FORMAT(cnt, ' Anzahl: %d')"/>
  <ForEachObject>
```

```
    <SetVar Name="v" BisAttribute="Object_Display_Kontext"/>
    <LogEntry Value="'Objekt: ' + v/>
  </ForEachObject>
</ReceiveAllGlobalNotify>
```

Beispiel zu 3

```
<SetVar Name="cnt" BisAttribute="COUNT"/>
<LogEntry Value="FORMAT(cnt, ' Anzahl: %d')"/>
<ForEachObject>
  <SetVar Name="v" BisAttribute="Object_Display_Kontext"/>
  <SetVar Name="vo" BisAttribute="bisB_ObjectImage"/>
  <LogEntry Value="'Objekt: ' + v + ' ' + vo"/>
</ForEachObject>
```

2.26 SetVar

Mit `<SetVar>` kann einer Variablen ein Wert zugewiesen werden. Jede Variable hat einen (eindeutigen) Namen. Die Variablen, welche so definiert werden können in <Variables>, bzw. <Var> wieder verwendet werden. Dies bietet eine Möglichkeit Werte umzuwandeln: Es wird eine Variable ‚a' geschrieben und definiert eine Variable ‚b', welche die Variable ‚a' verwendet, ohne dass diese explizit definiert ist.

`Name`:	Name der Variable (Mussfeld)
`Attribute` oder	
`BisAttribute`:	Attributbezeichner in BIS
`Expression`:	Ausdruck
`Value`:	Wert
`Format`:	definiert das Format, damit ist der Wert immer ein Text. ab Version 4.8.5: ‚AsInteger' gibt bei Aufzählungen den Integer, statt den Text zurück

Beispiele:

```
  <SetVar Name="msg" Value="Alles eingelesen!"/>
  <SetVar Name="anz" BisAttribute="COUNT"/>
  <SetVar Name="nam" BisAttribute="Object_Display_Kontext"/>
  <SetVar Name="bez" Expression="'Name' + nam"/>
  <SetVar Name="lst" />
  <SetVar Name="date" BisAttribute="bisB_CreationDate" Format="yyyymmdd"/>
  <SetVar Name="code" BisAttribute="bisB_2DBarcode" Format="base64"/>
  <SetVar Name="status" BisAttribute="bisP_Status" Format="AsInteger"/>
```

BisAttribute, Expression und Value werden alternativ angegeben. Mit BisAttribute wird das Attribut des 1. Objektes ausgewertet. Mit COUNT wird die Anzahl (integer) der Objekte wieder gegeben. Mit Expression kann ein Ausdruck verwendet werden, welcher sich auch auf Variablen beziehen kann. Mit Value kann ein konstanter Wert gesetzt werden. Wenn weder Expression noch BisAttribute noch Value angegeben werden, dann wird die Objektmenge in die Variable geschrieben. Diese Menge kann später mit <GetVar> wieder abgerufen werden (analog SAVEAS / RECALL in Filternavigation)

N.B.: `Value=""` hat dieselbe Bedeutung, wie Expression="''", also einen Leertext.

2.27 ShellExec / WinExec / ExecXDS / ExecBis

2.27.1 ShellExec Attribute:

File: Enthält die Datei als Ausdruck.

Wait: boolean („yes“ „no“)

TimeOut: integer; maximale Wartezeit in Millisekunden, wenn Wait = true ist. Null oder negative Werte bedeuten unendlich langes Warten. Default = 0. ab V4.9.8

KillProcessAfterTimeOut: boolean („yes“ „no“); bestimmt, ob der Prozess bei Wait = true und TimeOut > 0 automatisch beendet werden soll.
Default = false. ab V4.9.8

Der Text innerhalb des Elementes wird als Ausdruck ausgewertet und als Parameter verwendet.

```
<ShellExec File="'C:\Test.exe'"
           Wait="true"
           TimeOut="5000"
           KillProcessAfterTimeOut="true">'/PARAM1'</ShellExec>
```

2.27.2 WinExec Attribute:

File : Enthält das Programm und die Parameter als Ausdruck.

Der Text innerhalb des Elementes wird als Ausdruck ausgewertet und zusätzlich zum Wert von File angehägt. (Wait hat keine Wirkung, es wird immer auf das Prozessende des Programmes gewartet. Man beachte dass Programm und Dateien jeweils in „ Zeichen gesetzt werden sollten und nicht in , Beispiel:

```
<Variables>
  <Var Name="fil">"L:\Technische Dokumentation\XDS_Script.doc"</Var>
  <Var Name="dir">"C:\Programme\Microsoft Office\OFFICE11\WINWORD.EXE"</Var>
</Variables>
<Script>
  <WinExec File="dir + ' ' + fil" />
</Script>
```

Ab Version 5.1.2 wird ein ExitCode ungleich 0 als Fehler ausgewiesen und muss bei Bedarf mit OnError abgefangen werden. Der ExitCode ist mit einem Tab abgetrennt, kann also mit FIELD(varLastError, 1) aus der Meldung extrahiert werden.

2.27.3 ExecXDS Attribute:

Mit ExecXDS wird ein weiteres XDS ausgeführt. Die Logindaten werden dabei automatisch übergeben, vorausgesetzt das XDS selbst wurde schon mit codierten Logindaten aufgerufen. Siehe auch Funktion CODED in Funktionen in Ausdrücken

.

Name: Enthält den Namen (ID) des XDS Abgleiches (ohne XDS_ Präfix).

Wait: Boolean („yes“ „no“) – gibt an, ob auf den gestarteten Prozess gewartet werden soll. Vorgabe: no

SetVar: Expression– Damit kann eine Variable übergeben werden. Ab Version 4.10.7

Sessions:	Boolean (“yes” “no”) – Gibt an, ob das aufgerufene XDS Sessions verwenden soll. Vorgabe: die Einstellung des aktuellen Prozesses (vgl. XDS_IntroductionReference 1.3.4. Command line parameters). Ab Version 4.9.8
ReadOnly	Boolean (“yes” “no”) – Gibt an, ob das aufgerufene XDS die Datenbank nur lesend öffnen soll. Vorgabe: „no“. **NB**: `ReadOnly` schliesst die Verwendung von `Sessions` aus. Ist `ReadOnly` gesetzt, dann wird `Sessions` ignoriert. Ab Version 4.11.4

Beispiel, in dem zwei Variablen vName und vPath im Abgleich CADimport gesetzt werden:

```
<ExecXDS Name="CADimport"
         Wait="yes"
         Sessions="yes"
         SetVar="'vName=AA;vPath=' + vFn" />
```

2.27.4 ExecBis Attribute:

Name: Name der Operation gemäss unten stehender Tabelle

Wait: boolean („yes“ „no“) default = true

Führt eine SendToBIS Operation aus. Diese sind in

https://support.byron.ch/downloads/dokumente/ByronBIS_ActiveX_Doku.pdf beschrieben.

Aus der ActiveX Bezeichnung ist ersichtlich, ob der Befehl auf die Objektliste (sol) oder die Datenbank (sod) angewendet wird.

Wenn Wait = „no“ gesetzt ist, dann wird zum Befehlscode noch solAsync addiert.

Name hier im XDS	ActiveX Konstante
Open	solOpen
Select	solSelect
ShowInDocument	solShowInDocument
Actualize	solActualize
BringToFront	sodBringToFront
ActualizeAll	sodActualizeAll
ActualizeF5	sodActualizeF5
CloseBIS	sodCloseBIS
TerminateBIS	sodTerminateBIS

Achtung 1:

Wenn XDS blockierend aus BIS aufgerufen wird, muss diese Operation ‚nicht blockierend' aufgerufen werden: wait=“no“. Ansonsten blockieren sich BIS und XDS gegenseitig.

Achtung 2:

Man beachte, dass geänderte Werte erst nach der Transaktion für BIS sichtbar werden. Das bedeutet, dass ein Actualize in einer nachfolgenden Lesetransaktion gemacht werden muss:

```
<FetchError BisUpdate="yes">
  <Navigation>
    RECALL vCPUObject
    MODIFY MV_LastExport '=NOW'
  </Navigation>
</FetchError>
<FetchError BisUpdate="no">
  <Navigation>
    RECALL vCPUObject
  </Navigation>
  <ExecBis Wait="no" Name="Actualize" />
</FetchError>
```

2.28 ShowMessage

Zeigt einen Mittteilungsdialog. Dieser blockiert die weiter Ausführung, bis der Benutzer den Dialog bestätigt, bzw. beantwortet hat. Dieses Element darf deshalb nie in einer Update Transaktion verwendet werden.

ShowMessage kennt folgende Attribute:

Msg:	Enthält den Mitteilungstext als Ausdruck.
Type:	ein Wert aus: 'warning', 'error', 'information' oder 'confirmation'
Buttons:	Werte (kommagetrennt) aus: 'yes', 'no', 'ok', 'cancel', 'abort', 'retry', 'ignore', 'all', 'noToAll', 'yesToAll', 'help' Default , 'ok'
ResultVar	Name der Variablen, in welche die Zahl gespeichert wird, welche dem vom Benutzer gedrückten Knopf entspricht: ok = 1, cancel = 2, abort = 3, retry = 4, ignore = 5, yes = 6, no = 7, all = 8, noToAll = 9, , 'yesToAll'= 10

```
<ShowMessage Msg="FORMAT(logerr, '%d Fehler aufgetreten!') + #13 + #10
       + 'Soll die Protokolldatei angezeigt werden?'"
             Type="warning"
             Buttons="yes,no"
             ResultVar="rv"/>
```

2.29 Sleep

Mit Sleep wartet XDS die angegebene Zeit (nicht busy, d.h. ohne CPU auslastung). Während dieser Zeit können andere Prozesse aktiv sein. Die Wartezeit in Wait wird in Millisekunden angegeben. Der Wert in Wait ist eine Expression, dere Resultat muss eine ganze Zahl sein.

Warten für zwei Sekunden:

```
<Sleep Wait="2000" />
<!-- Da es eine expression ist könnte man das auch so schreiben: -->
<Sleep Wait="2 * 1000" />
```

2.30 SvgCreate (ab v5.2.1)

Mit SvgCreate kann ein Plan (.drw) in einen svg Plan konvertiert werden.

```
<SvgCreate Source = "vFile"/>
```

Es können folgende Attribute definiert werden:

`Source` (Expression)
Definiert die Quelldatei, dieses Attribut muss definiert werden.

`Target` (Expression)
Name der generierten SVG Datei.
Default: Gleicher Name wie Source, mit der Erweiterung SVG.

`Operate` (always, ifTargetOlder, ifTargetNotExists)
always: Die Konvertierung wird immer gemacht.
ifTargetOlder : Wird nur durchgeführt,, wenn die Quelldatei neuer ist als die Zieldatei.
ifTargetNotExists: Wir nur durchgeführt, wenn die Zieldatei nicht existiert.
Default: ifTargetOlder

`CreateSvgTag` (Boolean 0/1)
SVG Element erstellen
Default: 0

`CreateViewbox` (Boolean 0/1)
Viewbox erstellen
Default: 0

`CreateInitialDefinitions` (Boolean 0/1)
Initals DEFs erstellen
Default: 0

`CreateStrokeDefinition` (Boolean 0/1)
Stroke Definition erstellen
Default: 1

`OptimizeSpeed` (Boolean 0/1)
Geschwindigkeit optimieren
Default: 1

`UseGroupTransformations` (Boolean 0/1)
Eine Gruppe mit Transformation erstellen
Default: 0

`UseLayerVisibility` (Boolean 0/1)
Nur die sichtbaren Layer verwenden
Default: 1

2.31 (ab v5.3.1)

Mit WebMapDownload können Kacheln herunter geladen werden. Die Kacheln werden in ein Verzeichnis als PNG Dateien abgelegt. Diese Dateien werden dann als Cache vom Applikationselement Map verwendet.

Beispiel eines Aufrufes:

```
<WebMapDownload
      Style="BingHybrid"
      DirectoryName=" 'Z:\tiles' "
      MinLevel="8"     MaxLevel="10"
```

```
        Left="8.543"     Right="8.97"
        Top="47.355"     Bottom="47.02" />
```

2.31.1 Attribute

Style: Aufzählung: `WMS, BingSatellite, BingMap, BingHybrid, OSM, OSMstatic`
Details in Kap. 31.6 ElementMapService – style im Dokument:
https://support.byron.ch/downloads/dokumente/ByronBIS_ToolKonfiguration.pdf

Bottom: Float Expression *)
WGS84 Breitengrad der südlichen Begrenzung.

Clear: `Float` Default 0
Das maximale Alter, das eine Datei haben darf, ansonsten wird sie ersetzt durch den Download.
Beispiele:
-1: Nur für fehlende Dateien erfolgt ein Download
0: Alle Dateien und Verzeichisse werden gelöscht. (kompletter Download)
10: Alle Dateien älter als 10 Tage werden gelöscht.

DirectoryName: Text Expression Default: '%tmp%\kacheln'
Wurzelverzeichnuss für die Ablage. Achtung: Ein konstanter Wert muss in ' geschrieben werden, da es eine Expression ist.

Left: Float Expression *)
WGS84 Längengrad der westlichen Begrenzung.

MaxErrors: Integer Default 50
Anzahl fehlerhafte Kachel-Downloads, die toleriert werden, bevor der Befehl abgebochen wird.

MaxLevel: Integer oder Text Expression
Grösstes Zoom-Level.

MaxThreads: Integer Default 40
Eine kleinere Zahl spart Resourcen (Memory, gleichzeitige Verbindungen ins Internet).
Eine grössere Zahl erhöht die Geschwindigkeit.

MinLevel: Integer oder Text Expression
Kleinstes Zoom-Level.

Reaspect: `boolean`
Hier gibt es keinen Default, denn es gibt true, false oder weglassen.
Vgl. https://support.byron.ch/downloads/dokumente/ByronBIS_ToolKonfiguration.pdf

Retry: `integer` Default 3
Anzahl Versuche, eine Kachel zu downloaden.

Right: Float Expression *)
WGS84 Längengrad der östlichen Begrenzung.

SRS: Text Default "EPSG:4326" (WGS84)
Koordinaten System, wird nur für Style="WMS" verwendet.
http://spatialreference.org/ref/epsg/21781/

Top: Float Expression *)
WGS84 Breitengrad der nördlichen Begrenzung.

URL: Text
Wird nur für Style="WMS" verwendet.

Wait: Integer in Millisekunden. Default 0
Wartezeit nach Download einer Kachel.

*) Expressions für Längen und Breitengrade

Die Expression kann ein Float oder ein Text sein. Im Falle eines Textes wird dieser ohne Lokalisierung in einen Float verwandelt. Die folegenden 4 Expressions sind alle zulässig und identisch im Effekt:

Left="7.33" Left="'7.33'" Left="7 + 0.33" Left=" '7' + ' .33' "

2.31.2 Unterelemente Layers und Styles

Nur wenn WebMapDownload.Style="WMS".

<Layers> und <Styles> Details in Kap. 31.6 ElementMapService – WMS Kartenprovider im Dokument: https://support.byron.ch/downloads/dokumente/ByronBIS_ToolKonfiguration.pdf

Beispiel:

```
<WebMapDownload
      Style="WMS"      URL=http://wms.geo.admin.ch/
      MinLevel="8"     MaxLevel="10"
      Left="8.543"     Right="8.97"
      Top="47.355"     Bottom="47.02" >
  <Layers>
          ch.swisstopo.images-landsat25,
          ch.swisstopo.swissboundaries3d-gemeinde-flaeche.fill,
          ch.swisstopo.swissboundaries3d-bezirk-flaeche.fill,
          ch.swisstopo.swissboundaries3d-land-flaeche.fill,
          ch.swisstopo.swissboundaries3d-kanton-flaeche.fill
  </Layers>
  <Styles/>
</WebMapDownload>
```

2.32 WebMapTile

Mit WebMapTile werden aus einem Plan (drw-Datei) einzelne Kachlen (png-Dateien) erstellt.

Beispiel eines Aufrufes:

```
<WebMapTile
   ReadFile   = "vFile"
   TileDir    = "vTileDir"
   FloorLevel = "vLev"
   PlanRot    = "STRTOFLOAT(vR)"
   PlanDx     = "STRTOFLOAT(vX)"
   PlanDy     = "STRTOFLOAT(vY)"
   scale      = "0.01"
   DirDeep    = "3"
   Replace    = "ReplaceSmallerFile"
   KachelProBitmap = "2"
   Type       = "vTyp"
   ZoomLevel  = "vZlev"/>
```

Die Attribute:

`ReadFile`: Text Expression
Der Dateiname des Planes (DRW), welcher auf Kacheln gezeichnet wird.

`TileDir`: Text Expression
Der Name des Dateiordners, in welchem die PNG-Dateien erstellt werden. Die Dateien werden nicht direkt in diesem Ordner abgelegt, sondern es werden Unterordner erstellt. Siehe nachfolgende Tabelle Ordner- Dateistruktur.

`FloorLevel`: Text Expression
Stockwerk. Siehe nachfolgende Tabelle Ordner- Dateistruktur

`PlanRot`: Float Expression
Rotation des Plans im Bogenmass.

`PlanDx, PlanDy`: Float Expression
Verschiebung des Plans von seinen lokalen Koordinaten zu CH1903.

`scale`: Float Expression default = 1
1 wenn der Plan in Meter gezeichnet ist
0.01 wenn der Plan in Zentimeter gezeichnet ist
0.001 wenn der Plan in Millimeter gezeichnet ist

`DirDeep`: Integer Expression default = 3
steuert die Tiefe der Unterverzeichnisse, siehe Tabelle Ordnerstruktur der Dateiablage

`Replace`: Aufzählung default = `ReplaceSmallerFile`
`ReplaceFileNever` Ersetzt keine bestehenden Kacheln
`ReplaceSmallerFile` Ersetzt bestehende Kachel, wenn diese kleiner ist.
`ReplaceSmallerEqualFile` Ersetzt bestehende Kachel, wenn diese nicht grösser ist.
`ReplaceFileAlways` Ersetzt bestehenden Kachel immer.
`OverlayFile` Zeichnet auf bestehende Kachel.

`KachelProBitmap`: Integer Expression default = 0
0: Es wird eine (grosse) Bitmap generiert, welche den gesamten DRW-Plan umfasst. Diese Bitmap wird dann in einzelne Kacheln (256x256 png Datei) zerlegt. Dies ist performant, kann aber zu Fehlern führen, wenn die Bitmap zu gross wird.
1: Es wird jeweils eine Bitmap mit 4 Kacheln erzeugt.
2: Es wird jeweils eine Bitmap mit 16 Kacheln erzeugt.
n: Es wird jeweils eine Bitmap mit 4*n*n Kacheln erzeugt.

`Type`: Text Expression
Typ des Plans, Siehe nachfolgende Tabelle Ordner- Dateistruktur.

`ZoomLevel`: Integer Expression
Google Zoom-Level Siehe nachfolgende Tabelle Ordner- Dateistruktur

Ordnerstruktur der Dateiablage Beispiele:

TileDir	FloorLevel	ZoomLevel	Type	DirDeep	PNG-Ablage
'C:\Temp\K'	'5'	'18'	'Arch'	3	'C:\Temp\K\L5\18\Arch\x_y.png'
'C:\Temp\K'	'5'	'18'	'Arch'	2	'C:\Temp\K\L5\18\Arch_x_y.png '
'C:\Temp\K'	'5'	'18'	'Arch'	1	'C:\Temp\K\L5\18_Arch_x_y.png '
'C:\Temp\K'	'5'	'18'	'Arch'	0	'C:\Temp\K\L5_18_Arch_x_y.png '

2.33 WHILE

Mit WHILE werden die enthaltenen Elemente so lange ausgeführt bis die Condition false ergibt.

Das Attribut Condition enthält eine boolsche Bedingung

```
<SetVar Name="j" Expression="0"/>
<WHILE Condition="j &lt; 5000">
  <SetVar Name="j" Expression="j + 1"/>
</WHILE>
```

Die Anzahl der Wiederholungen ist nur durch die Condition gesteuert und unabhängig von der Anzahl der Objekte.

2.34 WriteXML / XMLelement / XMLtext / XMLattribute / XMLcomment

Eine XML Datei wird mit WriteXML geschrieben. Innerhalb dieses Elementes ist die Datei geöffnet und kann durch XMLelement, XMLtext und XMLattribute beschrieben werden.

Durch Einbezug von Navigation und SetVar Elemente können Daten aus ByronBIS in eine XML Datei geschrieben werden.

Beispiel einer (statischen) XML Datei.

```
<Variables>
  <Var Name="fx" Value="=OSENV('temp') + '\aa.xml'" />
</Variables>

<Script>
  <WriteXML RootName="Hallo" FileName="fx">
    <XMLattribute Name="'xmlns:xsi'"
                  Value="'http://www.w3.org/2001/XMLSchema-instance'"/>
    <XMLelement Name="'ab'">
        <XMLtext Value="'Ein Text ' + FORMAT(NOW, 'ddd.mmm.yyy hh.nn.sss')"/>
        <XMLattribute Name="'ID'" Value="12"/>
        <XMLelement Name="'cd'"/>
      </XMLelement>
  </WriteXML>
  <LogEntry Value="'geschrieben: ' + fx"/>
</Script>
```

Wenn XMLelement als 1. Element ein <IterateNav> Element mit einer Filter-Navigation enthält, dann wird das XMLelement iterativ angewendet. Beispiel:.

```
<Variables>
  <Var Name="fx" Value="=OSENV('temp') + '\Org.xml'" />
</Variables>
<Script>
  <WriteXML RootName="Organisationen" FileName="fx">
    <Navigation>
      START CLASS bisB_OrganisationContainerRoot
      VIA Object_To_Children CLASS bisB_Organisation
    </Navigation>
    <ForEachObject>
      <XMLelement Name="'Organisation'">
        <IterateNav>
```

```
          VIA Object_To_Children
          CLASS BisB_OrganisationUnit
        </IterateNav>
        <SetVar Name="v" BisAttribute="Object_Display_Kontext"/>
        <XMLattribute Name="'name'" Value="v"/>
      </XMLelement>
    </ForEachObject>
 </WriteXML>
</Script>
```

Man beachte, dass XMLelement innerhalb eines ForEachObject steht, damit das Element für jedes Objekt ausgeführt wird. Die iterative Anwendung durch IterateNav impliziert ein ‚ForEachObject'. Das Resultat sieht damit so aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Organisationen>
  <Organisation name="Byron Informatik AG">
    <Organisation name="Entwicklung">
      <Organisation name="Lernende"/>
    </Organisation>
    <Organisation name="Geschäftsführung"/>
    <Organisation name="Verwaltung"/>
    <Organisation name="Marketing"/>
  </Organisation>
  <Organisation name="TripleW AG">
    <Organisation name="F+E"/>
    <Organisation name="Administration"/>
    <Organisation name="Marketing"/>
  </Organisation>
</Organisationen>
```

2.34.1 WriteXML

`Id`	Default "" Dieser Wert muss dann von den Operationen verwendet werden. Gleiches Konzept wie bei OpenFile.
`RootName`	text: Root Elementname des Dokuments
`FormattedOutput`	boolean: default "no" >= V 4.4
`FileName`	expression: Dateiname
`encoding`	string: default "ISO-8859-1"

2.34.2 XMLelement

`Id`	Default "" Damit definiert man auf welche Datei sich der Befehl bezieht.
`Name`	expression: Name des XML Element.
`Value`	expression: Falls gesetzt, wird die ausgewertete Expression als Text in das Element eingesetzt.
`BisAttribut`	string: Bis-Attribute der Wert wird als Text in das Element eingesetzt. (vgl. Format)
`Format`	Wird für BisAttribut eingesetzt.

Kann weitere Elemente enthalten. Wenn das erste Element <IterateNav> ist, dann enthält dieses eine Filter-Navigation, die iterativ angewendet wird, bis kein Objekt mehr erreicht wird. Ohne <IterateNav> wird der Inhalt von XMLelement genau einmal mit der aktuellen Objektmenge ausgeführt.

2.34.3 XMLattribute

`Id`	Default "" Damit definiert man auf welche Datei sich der Befehl bezieht.
`Name`	expression: Name des XML Attributes.
`Value`	expression: Wert des Text Attributes. Typisierte Werte werden gemäss XSD Standard ausgegeben, z.B. Boolean: ‚true' / ‚false' Datum: , 2009-05-05T15:25:48' Null: Attribut wird NICHT geschrieben, bzw. entfernt. Floats: immer mit ‚.' (Punkt) unabhängig von der Lokalisierung. Falls dies nicht passend ist, muss der Wert mit einem entsprechenden Format in eine Variable geschrieben werden .
`BisAttribute`	string: Attributname in BIS. Dies ersetzt das Value Attribut. Die ist eine Abkürzung, mit der ein SetVar eingespart werden. <SetVar Name="vProjectID" BisAttribute="bisX_ProjectID"/> <XMLelement Name="'Project-ID'"> <XMLattribute Name="'ID'" Value="vProjectID"/> abgekürzt: <XMLelement Name="'Project-ID'"> <XMLattribute Name="'ID'" BisAttribute="bisX_ProjectID"/> vgl. <SetVar BisAttribute
`Format`	Wird für BisAttribut eingesetzt.

2.34.4 XMLtext

`Id`	Default "" Damit definiert man auf welche Datei sich der Befehl bezieht.
`Value`	expression: Wert des Text Nodes im aktuellen Element.
`AsXML`	boolean default "no" Wenn „yes“: dann ist der Text (Wert) selbst ein XML Text, d.h. der Wert wird als ‚Subtree' angehängt. Ansonsten („no“ werden ‚<' und andere Zeichen ersetzt. (ab v 4.4)
`NilFlag`	boolean default "no" steuert ob bei null-Werten ein Attribut `xsi:nil="true"` an das Element geschrieben werden soll. (ab v4.4)
`BisAttribute`	vgl. XMLattribute.
`Format`	vgl. XMLattribute.

Beispiele:

```
<SetVar Name="v" BisAttribute="bisB_ObjectImage"/>
<XMLtext Value="v"/>
<XMLtext BisAttribute="bisB_2DBarcode" Format="base64"/>
<XMLtext BisAttribute="bisB_CreationDate" Format="yyyymmdd"/>
```

2.34.5 XMLcomment (ab v4.6)

`Id`	Default "" Damit definiert man auf welche Datei sich der Befehl bezieht.
`Value`	expression: Kommentar als Attribut (alternativ zu Text)

`Text` string: Kommentar als Text Knoten (alternativ zu Attribut Value)

Beispiele:

```
<XMLcomment Value="'generated by Byron/BIS'"/>
<XMLcomment>generated by Byron/BIS</XMLComment>
```

2.34.6 XMLProcessingInstructions (ab v4.11.2)

`Id` Default "" Damit definiert man auf welche Datei sich der Befehl bezieht.

`Target` expression: Instructiontyp, z.B. 'xml-stylesheet '

`Data` expression: Attribute, z.B. type="text/xsl" href="pq.xsl"

Beispiel:

```
<XMLProcessingInstructions Target="'xml-stylesheet'"
                           Data="'type='+#34+'text/xsl'+#34+'
                           href='+#34+'pq.xsl'+#34"/>
```

erzeugt

```
<?xml-stylesheet type="text/xsl" href="pq.xsl"?>
```

N.B. Im Gegensatz zu XMLelement, XMLcomment, XMLText etc. fügt sich die Instruction nicht an der Stelle ein, an der sie definiert ist, sondern eine Stufe höher, vor dem Element in dem es definiert ist.

2.35 CalDavConnection / CalDavForEachServerEvent / CalDavPushEvent / CalDavDeleteEvent / CalDavReadEventDetails / CalDavFetchServerEvents / CalDavEncodeDateTime / CalDavDecodeDateTime / CalDavGetRecurrenceInstances

Mit CalDavConnection wird eine Verbindung zu einem CalDav-Server aufgebaut, um Kalendereinträge zu synchronisieren.

Der CalDav Standard bestimmt, dass Einträge auf dem Server über einen „Dateinamen" identifiziert werden. Der Etag eines Termins ändert sich mit jeder Änderung auf dem Server und dient somit zum schnellen Überprüfen, ob sich ein Termin verändert hat.

Achtung, die Unique-Id, die ein Termin in iCal haben sollte, hat keinen Einfluss auf CalDav und muss für die grundlegende Funktion von CalDav nicht mal gesetzt sein. Allerdings muss auch diese UID gesetzt und eindeutig sein, damit zum Beispiel Einladungen korrekt funktionieren. Ausserdem sehen manche Dateinamen aus, wie eine GUID, die kann, aber muss nicht gleich sein wie die UID in iCal.

Für eine komplette Synchronisation muss man in der Regel:

1. Alle Termine ausfindig machen, die sowohl lokal, als auch auf dem Server verändert wurden, und die Konflikte auflösen oder markieren. Dabei Kombinationen von verändert und gelöscht berücksichtigen, z.B. lokal geändert und auf dem Server gelöscht.
2. Alle auf dem Server gelöschten Termine lokal löschen.
3. Alle lokal geänderten und neu erstellten Termine an den Server schicken.
4. Alle lokal gelöschten vom Server löschen.
5. Alle auf dem Server neuen oder geänderten Termin holen und lokal anpassen.

Ein Beispiel findet sich hier.

2.35.1 CalDavConnection

Host: Der Adresse des Kalender-Hosts.

CalendarLocation: Der Pfad des Kalenders auf dem oben gegebenen Host.

Username:	Benutzername
Password:	Passwort

2.35.2 CalDavForEachServerEvent

Liest alle Termine aus dem Kalender, die dem Filter in SearchFilter entsprechen.

Bemerkung: Die XML-Namespace-Präfixe sind in XDS mit Kleinbuchstaben implementiert.

Bemerkung: Falls ein vorher bestehender Filter restriktiver gemacht wird, oder ein neuer Filter eingeführt wird, werden beim obigen Skript lokal geänderte Termine, die neuerdings ausgefiltert werden als Synchronisationsproblem markiert, da sie aus Client-Sicht auf dem Server gelöscht und lokal geändert wurden.

FileNameVar:	Variable, in die der Dateiname des CalDav-Termins geschrieben wird.
EtagVar:	Variable, in die der Etag des CalDav-Termins geschrieben wird.
SearchFilter:	XML-String, der als Filter an den CalDav-Server geschickt wird. Siehe Beispiel oder http://tools.ietf.org/html/rfc4791#section-9.7

2.35.3 CalDavPushEvent

Schreibt einen Termin auf den Server. Falls der Termin dort schon vorhanden ist, wird er überschrieben.

FileName	Der Dateiname, unter dem der Termin gespeichert wird auf dem Server.
ICalString	String, mit dem Termin im iCal-Format.
EtagVar	Variable, in die der geänderte Etag gespeichert wird, der vom Server zurückkommt.

2.35.4 CalDavDeleteEvent

Löscht einen Termin vom Server.

FileName	Der Dateiname, unter dem der Termin gespeichert ist auf dem Server.

2.35.5 CalDavReadEventDetails

Liest einen Kalendereintrag, den CalDavFetchServerEvents geladen hat, datensatzweise aus.

Die Datensätze werden aufgeteilt in die Bezeichnung, Parameter und den Inhalt und jeweils in die Variablen von RecNameVar, ParametersVar und RecValueVar geschrieben.

Beispiel:
„SUMMARY:Firmenrundgang“
wird aufgeteilt in
RecNameVar=“SUMMARY“, RecValueVar=“Firmenrundgang“, ParametersVar=““

„SUMMARY;LANGUAGE=de-de:Firmenrundgang„
wird aufgeteilt in
RecNameVar=“SUMMARY“, ParametersVar=“LANGUAGE=de-de“, RecValueVar=“Firmenrundgang“

RecNameVar	Variable, die die jeweilige Bezeichnung des iCal-Datensatzes enthält.
RecValueVar	Variable, die den jeweiligen Inhalt des iCal-Datensatzes enthält.
ParametersVar	Variable, die Parameter zum iCal-Datensatz enthält.

2.35.6 CalDavFetchServerEvents

Liest viele vollständige Kalendereinträge vom Server passend zu dem/den gegebenen Dateinamen.

Man kann die Methode entweder mit einem einzigen Dateinamen in ‚FileName' oder mehreren Tab-getrennten Dateinamen in FileNames aufrufen.

FileName	Ein einzelner Dateiname.
FileNames	Tab-getrennte Liste aller Dateinamen, die FetchServerEvents holen soll.
FileNameVar	Variable, die bei der Iteration den gegenwärtigen Dateinamen enthält.
ExpandRecurringFrom	Falls hier und bei ‚ExpandRecurringUntil' ein Datum steht, werden Serientermine von diesem Datum an bis ‚ExpandRecurringUntil' expandiert.
ExpandRecurringUntil	Falls hier und bei ‚ExpandRecurringFrom' ein Datum steht, werden Serientermine von ‚ExpandRecurringFrom' bis zu diesem Datum expandiert.
ClientExpansion	Boolean Expression, die definiert, ob Serientermine client- oder serverseitig expandiert werden sollen. Für clientseitige Expansion müssen die Grenzen mit den obengenannten Attributen definiert werden.

2.35.7 CalDavEncodeDateTime

Formatiert einen Datum- und Zeitstring ins iCal-Format der Form yyyymmdd'T'hhmmss'Z', der zeitzonenunabhängig ist.

EncodedTime	Enthält die Variable, in die der formatierte String gespeichert wird.
DecodedTime	Expression, die die zu formatierende Zeit enthält.
AllDay	Boolean Expression, die angibt, ob es sich um einen ganztägigen Termin handelt, dessen Datumsstring anders formatiert wird.
KeyStr	String Expression, die die Bezeichnung der iCal-Eigenschaft enthält. Z.B: „DTSTART". Der iCal-Datensatz wird abhängig davon, ob es sich um einen ganztägigen Termin handelt, anders formatiert.

2.35.8 CalDavDecodeDateTime

Liest einen Datum- und Zeitstring vom iCal-Format aus unter Berücksichtigung der im String eingebetteten Zeitzone.

EncodedTime	Expression, die die auszulesende Zeit enthält.
DecodedTime	Enthält die Variable, in die die ausgelesene Zeit gespeichert wird.
Parameters	Enthält die Parameter, die zur Zeit gehören. Z.B. „TZID=Europe/Amsterdam"
AllDay	Name einer Variablen, die nach dem Auslesen geprüft werden kann, ob es sich um einen ganztägigen Termin handelt.

2.35.9 CalDavGetRecurrenceInstances

Gibt die einzelnen Daten von wiederkehrenden Terminen zurück. Manche Termine sind nicht durch eine Endzeit, sondern durch eine Dauer beschrieben. In dem Fall ist HasEndDateTime=false und EndDateTimeVar hat einen undefinierten Wert. Die Dauer kann vom Termin, zu dem diese Serientermin-Instanz gehört abgelesen werden.

StartDateTimeVar	Variable, in die die Anfangszeit einer Serientermin-Instanz geschrieben wird.

EndDateTimeVar	Variable, in die die Endzeit einer Serientermin-Instanz geschrieben wird.
HasEndDateTime	Boolean Variable, die gesetzt wird, je nachdem, ob eine Serientermin-Instanz eine Endzeit besitzt.
RecordVar	Variable, in die der ganze iCal-Text des Serientermins geschrieben wird (bei serverseitiger Expansion). In manchen Fällen kann dies hilfreich sein.

2.36 CardDavConnection / CardDavFetchCards / CardDavReadCard / CardDavWriteCard

Mit CardDavConnection wird eine Verbindung zu einem CardDav-Server aufgebaut, um Adressbücher und Kontakte zu synchronisieren.

Der CardDav Standard bestimmt, dass Einträge auf dem Server über einen „Dateinamen" identifiziert werden. Der Etag eines Kontakts ändert sich mit jeder Änderung auf dem Server und dient somit zum schnellen Überprüfen, ob sich ein Kontakt verändert hat.

Achtung, die Unique-Id, die ein Kontakt in vCard haben sollte, dient nicht zum Identifizieren der Kontakte auf dem Server. Häufig sehen sowohl der Detainame, wie auch die Unique-Id aus wie eine GUID und können, aber müssen nicht gleich sein.

2.36.1 CardDavConnection

Host:	Der Adresse des Adressbuch-Hosts.
AddressbookLocation:	Der Pfad des Adressbuchs auf dem oben gegebenen Host.
Username:	Benutzername
Password:	Passwort

2.36.2 CardDavFetchCards

Liest alle Kontakte aus dem Adressbuch.

FileNameVar:	Variable, in die der Dateiname des CardDav-Kontakts geschrieben wird.
EtagVar:	Variable, in die der Etag des CardDav-Kontakts geschrieben wird.
FetchCardContents:	Boolean Expression, die angibt, ob die ganzen Kontaktinhalte geladen werden sollen, oder nur eine Liste aller Dateinamen und Etags. Falls sich nur wenige Kontakte pro Synchronisation ändern, sollte hier FALSE gesetzt werden. (Die Inhalte der Kontakte müssen dann mit CardDavReadCard individuell nachgeladen werden.)

2.36.3 CardDavWriteCard

Schreibt einen Kontakt auf den Server. Falls der Kontakt dort schon vorhanden ist, wird er überschrieben.

FileName	Der Dateiname, unter dem der Kontakt gespeichert wird auf dem Server.
VCardString	String, mit dem Kontakt im vCard-Format.
EtagVar	Variable, in die der geänderte Etag gespeichert wird, der vom Server zurückkommt.

2.36.4 CardDavReadCard

Dieses Element gibt es als Kind-Element zu CardDavFetchCards und alleinestehend. Wenn FileName gesetzt ist, wird ein vollständiger Kontakt vom Server passend zum gegebenen Dateinamen gelesen.

Wenn das Element in einem CardDavFetchCards verschachtelt ist, dient es zum Auslesen der einzelnen Kontakte, die FetchCards geholt hat.

Über die erhaltenen Kontakte im vCard-Format wird Datensatzweise iteriert.

Die Datensätze werden aufgeteilt in die Bezeichnung, Parameter und den Inhalt und jeweils in die Variablen von RecNameVar, ParametersVar und RecValueVar geschrieben.

Beispiel:
„SUMMARY:Firmenrundgang“
wird aufgeteilt in
RecNameVar=“SUMMARY“, RecValueVar=“Firmenrundgang“, ParametersVar=““

„SUMMARY;LANGUAGE=de-de:Firmenrundgang„
wird aufgeteilt in
RecNameVar=“SUMMARY“, ParametersVar=“LANGUAGE=de-de“, RecValueVar=“Firmenrundgang“

FileName	Der Dateiname, unter dem der Kontakt gespeichert ist auf dem Server.
RecNameVar	Variable, die die jeweilige Bezeichnung des vCard-Datensatzes enthält.
RecValueVar	Variable, die den jeweiligen Inhalt des vCard-Datensatzes enthält.
ParametersVar	Variable, die Parameter zum vCard-Datensatz enthält.

2.36.5 CardDavDeleteCard

Löscht einen Kontakt vom Server.

FileName	Der Dateiname, unter dem der Kontakt gespeichert ist auf dem Server.

3 Anwendung

Für detaillierte Informationen über ein Script Element siehe Script Elemente

3.1 Objekte aus Textdatei erzeugen

Voraussetzung:

Eine Tab getrennte Textdatei im %Temp% Verzeichnis: Personen.txt
Vor und Nachname sind durch einen Tabulator getrennt.

Für jede Zeile wird eine Person im Default-Container der Personen angelegt.

Die neu angelegten Personen erhalten eine Telefonnummer ‚-0-'. Alle Personen mit dieser Telefonnummer werden jeweils zuvor gelöscht.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="fn" Value="=OSENV('Temp') + '\Personen.txt'" />
    <Var Name="fTempFlag" Value="='-0-'" />
    <Var Name="fVorname" Value="=FIELD(rec, 0)" />
    <Var Name="fNachname" Value="=FIELD(rec, 1)" />
  </Variables>

  <Script>
    <LogEntry Value="'lese: ' + fn" />
    <!-- 'alte' Daten löschen -->
    <Navigation>
      START INSTANCES Person
      VALUE bisB_PhoneNumber = DATAOF fTempFlag
      DELETE
    </Navigation>
    <SetVar Name="cnt" Expression="0" />
    <OpenFile Mode="read" FileName="fn" RecVar="rec">
      <SetVar Name="cnt" Expression="cnt + 1" />
      <!-- irgend EIN Objekt, danach Create
           rec ist gesetzt, fVor-, fNachname beziehen sich auf rec -->
      <Navigation>
        START CLASS Space
        CREATE Person ()
        MODIFY bisB_Firstname DATAOF fVorname
        MODIFY bisB_Lastname DATAOF fNachname
        MODIFY bisB_PhoneNumber DATAOF fTempFlag
      </Navigation>
    </OpenFile>
    <LogEntry Value="'fertig: ' + fn" />
    <LogEntry Value="'Anzahl Records ' + FORMAT(cnt,'%d')" />
  </Script>
</Defs>
```

3.2 Textdatei lesen/schreiben

Dieses Script erzeugt eine Muster-Datei. Diese kann als rlage gebraucht werden um eingelesen zu werden. Die Dateinamen sind aus dem Script ersichtlich.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="dir" Value="= OSENV('ALLUSERSPROFILE') + '\ByronBIS\DemoDb\XDS\'" />
    <Var Name="fno" Value="= dir + 'PersonenMuster.txt'" />
    <Var Name="fni" Value="= dir + 'Personen.txt'" />
    <Var Name="tab" Value="= #9" />
  </Variables>

  <Script LogToSysLog="TRUE">
    <!-- eine MusterDatei schreiben in Lesetransaktion -->
    <FetchError BisUpdate="no">
      <Navigation>
        START VALUE bisB_PersonalNumber '*'
        COUNT (VIA bisB_PersonToOrgUnit) = 1
        COUNT (VIA Room_Of_Component) = 1
        SORT (+ bisB_ObjectImage) PICK 0 9
      </Navigation>
      <LogEntry Value="'schreibe Datei: ' + fno" />
      <OpenFile Mode="create" FileName="fno">
        <!-- Titel schreiben -->
        <SetVar Name="vNum" Value="Personalnummer" />
        <SetVar Name="vNam" Value="Nachname" />
        <SetVar Name="vVor" Value="Vorname" />
        <SetVar Name="vTel" Value="Telefon" />
        <SetVar Name="vOrg" Value="Organisation" />
        <SetVar Name="vRoo" Value="Raum" />
        <SetVar Name="rec"
                Expression="vNum +tab+ vNam +tab+ vVor +tab+ vTel +tab+ vOrg +tab+
vRoo" />
        <Next Variable="rec" />
        <ForEachObject>
          <!-- Variablen setzen -->
          <SetVar Name="vObjPerson"/>
          <SetVar Name="vNum" Attribute="bisB_PersonalNumber" />
          <SetVar Name="vNam" Attribute="bisB_Lastname" />
          <SetVar Name="vVor" Attribute="bisB_Firstname" />
          <SetVar Name="vTel" Attribute="bisB_PhoneNumber" />
          <Navigation>
            VIA bisB_PersonToOrgUnit
          </Navigation>
          <SetVar Name="vOrg" Attribute="Name" />
          <Navigation>
            RECALL vObjPerson
            VIA Room_Of_Component
          </Navigation>
          <SetVar Name="vRoo" Attribute="bisB_SpaceId" />
          <!-- Record setzen -->
          <SetVar Name="rec"
```

```
                  Expression="vNum +tab+ vNam +tab+ vVor +tab+ vTel +tab+ vOrg
+tab+ vRoo"/>
          <!-- Record schreiben -->
          <Next Variable="rec" />
        </ForEachObject>
      </OpenFile>
    </FetchError>

    <IF Condition="FILEAGE(fni) &lt; 0">
      <ShowMessage Msg="'Datei wird nicht eingelesen, da es sie nicht gibt'
                        + #13 + #10 + fni"
                   Type="information" Buttons="ok" />
    <ELSE/>
      <!-- eine Datei einlesen -->
      <LogEntry Value="'lese Datei: ' + fno" />
      <SetVar Name="zz" Expression="0"/> <!-- Zeilenzähler -->
      <FetchError BisUpdate="yes">
        <OpenFile Mode="read" FileName="fni" RecVar="rec">
          <SetVar Name="zz" Expression="zz+1"/>
          <IF Condition="zz > 1"> <!-- Titelzeile ignorieren -->
            <SetVar Name="vNum" Expression="FIELD(rec,0)" />
            <SetVar Name="vNam" Expression="FIELD(rec,1)" />
            <SetVar Name="vVor" Expression="FIELD(rec,2)" />
            <SetVar Name="vTel" Expression="FIELD(rec,3)" />
            <SetVar Name="vOrg" Expression="FIELD(rec,4)" />
            <SetVar Name="vRoo" Expression="FIELD(rec,5)" />
            <Navigation>
              [
                START VALUE bisB_PersonalNumber = DATAOF vNum
              ELSIF = 0
                CREATE 1 Person ()
                MODIFY bisB_PersonalNumber DATAOF vNum
              ]
              MODIFY bisB_Lastname DATAOF vNam
              MODIFY bisB_Firstname DATAOF vVor
            </Navigation>
            <SetVar Name="vObjPerson" />
            <IF Condition="STRLEN(vTel) > 0">
              <Navigation>
                MODIFY bisB_PhoneNumber DATAOF vTel
              </Navigation>
            </IF>
            <IF Condition="STRLEN(vOrg) > 0">
              <Navigation>
                START INSTANCES bisB_OrganisationUnit
                VALUE Name = DATAOF vOrg
                [
                ELSIF = 1
                  SAVEAS orgunt
                  RECALL vObjPerson
                  BIND bisB_PersonToOrgUnit (RECALL orgunt) REBIND
                ]
              </Navigation>
            </IF>
```

```
            <Navigation>
              RECALL vObjPerson
              BIND Room_of_Component (
                START VALUE bisB_SpaceId = DATAOF vRoo
              ) REBIND
            </Navigation>
          </IF>
        </OpenFile>
      <OnError Transaction="abortAtBegin"/>
        <LogEntry Type="warning"
           Value="'Fehler beim Einlesen der Zeile: ' + FORMAT(zz, '%d')" />
      </FetchError>
    </IF>
  </Script>
</Defs>
```

3.3 MS-EXCEL – Datei exportieren (Sylk)

Werden MS-EXCEL-Dateien als Textdateien erzeugt (Tab-getrennt oder CSV), dann ergibt sich oft das Problem, dass EXCEL Texte als Zahlen interpretiert. Eine Lösung für dieses Problem ist die Verwendung von SYLK-Dateien.

Die Abkürzung SYLK steht für SYmbolik LinK und ist ein von Microsoft entwickeltes Dateiformat (.slk) für die Datenübermittlung zwischen spreadsheets. Weil Microsoft Excel .slk öffnen und als gewöhnliche Excel-Tabellen darstellen kann, ist es möglich mit Hilfe eines XDS-Scripts aus Datenbankinhalten, wie sie in Byron BIS enthalten sind, eine .slk Datei zu generieren und somit in ein für Excel verständliches Format zu exportieren.

Um eine .slk- Datei zu generieren muss man mit Hilfe der Filternavigation im XDS-Script über die gewünschten Elemente iterieren und die jeweiligen Werte im entsprechenden Sylk-Format in die neue Datei schreiben. Ein Beispiel dazu ist weiter unten verfügbar.

Achtung: Sylk-Dateien unterstützen keine Farben! Weder Textfarben noch Hintergrundfarben. Um Farben zu verwenden sollte das XML-Format zum Exportieren gewählt werden (siehe Kapitel 3.4).

Beispiel: die folgende Tabelle

023	234
023	234
023	'234'
023	Hallo

Sieht als SYLK-Datei aus wie folgt:

```
ID;PXDS
P;P@
F;C1;P0
F;C2;P0
C;Y1;X1;K"023"
C;Y1;X2;K234
C;Y2;X1;K"023"
C;Y2;X2;K234
C;Y3;X1;K"023"
C;Y3;X2;K"'234'"
C;Y4;X1;K"023"
C;Y4;X2;K"Hallo"
E
```

Bemerkungen zum SYLK-Format (vgl. Wikipedia)

`ID;PXDS` – Header des SYLK-Formates (P gibt das Programm an, in diesem Fall XDS)

`P;P@` - Definition des vordefinierten Formats mit Index 0. Formatdefinition gemäss EXCEL (hier Text, es ginge aber z.B. auch `P;P#,##0`).

`F;C1;P0` – Zuweisung des Formats mit Index 0 an die Spalte 1 – Formate können aber auch auf Zellenebene zugewiesen werden (mit ;Xn;Yn).

`F;C2;P0` – Zuweisung des Formats mit Index 0 an die Spalte 2.

C;Y1;X1;K"023" – Zuweisen von Daten zu einer Zelle (;Y1;X1).
Achtung: Texte müssen mit doppelten Anführungsstrichen eingefasst werden. Wie doppelte Anführungsstriche verarbeitet werden (escaping) ist unklar (ungültig: K"Hal"lo"). **Mehrzeilige** Texte können

exportiert werden, indem statt „#13#10“ (CR LF) „#27 :“ (ESC space Doppelpunkt) in die Datei geschrieben wird (vgl. Bsp – Variable f1m).

E – der letzte Record

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="fo" Value="= 'C:\temp\Erstellerkatalog.slk' "/>
    <Var Name="f1" Value="Bezeichnung"/>
    <Var Name="f1m" Value="=STRREPL(f1, #13 + #10, #27 + ' :')"/>
      <!-- escaping für mehrzeilige Werte -->
    <Var Name="f2" Value="Ersteller"/>
  </Variables>

  <Script>
    <FetchError BisUpdate="no">
      <OpenFile Id="out" Mode="create" FileName="fo">
        <!-- "Header" der Datei -->
        <SetVar Name="vData" Value="ID;PXDS"/>
        <Next Id="out" Variable="vData"/>
        <!-- Formatdefinition Text -->
        <SetVar Name="vData" Value="P;P@"/>
        <Next Id="out" Variable="vData"/>
        <!-- Zuweisung des Formats an Spalte 1 -->
        <SetVar Name="vData" Value="F;C1;P0"/>
        <Next Id="out" Variable="vData"/>
        <!-- Zuweisung des Formats an Spalte 2 -->
        <SetVar Name="vData" Value="F;C2;P0"/>
        <Next Id="out" Variable="vData"/>
        <!-- Titelzeile -->
        <SetVar Name="vData" Expression="'C;Y1;X1;K' + #34 + f1 + #34"/>
        <Next Id="out" Variable="vData"/>
        <SetVar Name="vData" Expression="'C;Y1;X2;K' + #34 + f2 + #34"/>
        <Next Id="out" Variable="vData"/>

        <SetVar Name="vLine" Expression="2"/>

        <Navigation>
          START INSTANCES BBI_ErstellerKatalog
          VIA Object_To_Children
          CLASS BBI_Ersteller SORT(+BBI_Erstellerkuerzel)
        </Navigation>
        <ForEachObject>
          <SetVar Name="vLineStart" Expression="FORMAT(vLine,'C;Y%d;')" />
          <SetVar Name="f1" BisAttribute="Name"/>
          <SetVar Name="vData" Expression="vLineStart + 'X1;K' + #34 + f1m + #34"/>
          <Next Id="out" Variable="vData"/>

          <SetVar Name="f2" BisAttribute="BBI_Erstellerkuerzel"/>
          <SetVar Name="vData" Expression="vLineStart + 'X2;K' + #34 + f2 + #34"/>
          <Next Id="out" Variable="vData"/>

          <SetVar Name="vLine" Expression="vLine +1"/>
        </ForEachObject>
```

```
        <!-- "Footer" der Datei -->
        <SetVar Name="vData" Value="E"/>
        <Next Id="out" Variable="vData"/>
      </OpenFile>
    </FetchError>
  </Script>
</Defs>
```

3.4 MS-EXCEL – Datei exportieren (XML)

Ein Microsoft Excel-Dokument kann neben herkömmlichen Dateiformaten wie .xls und .xlsx auch im XML-Dateiformat (.xml) gespeichert werden. Damit eine XML-Datei automatisch als Exceldatei erkannt wird, muss die vordefinierte Process-Instruction

```
<?mso-application progid="Excel.Sheet"?>
```

gleich nach dem XML-Header vorkommen. Microsoft bietet zur Erstellung von Exceldateien im XML-Format eine Spreadsheet-Reference an. Diese Spreadsheet-Reference gibt vor, wie ein XML-Dokument aufgebaut sein muss. So ist zum Beispiel definiert, welche Subknoten ein Tag haben muss oder kann und welche Otionen durch Attribute verfügbar sind. Mit Hilfe der dazugehörigen Namespaces von Excel `"urn:schemas-microsoft-com:office:excel"` und dem Spreadsheet `"urn:schemas-microsoft-com:office:spreadsheet"`, können einige der Funktionalitäten, wie man sie aus Excel kennt, in ein XML-Dokument geschrieben werden. Dazu muss ein XDS-Script geschrieben werden, welches ein XML-Dokument nach den vorgeschrieben Regeln generiert, welches durch doppelklicken automatisch in Excel geöffnet wird und dort wie gewohnt bearbeiten kann. Ein Beispiel für ein XDS-Script und ein Beispiel für ein minimales XML-Excel-Dokument sind weiter unten in *Beispiele* zu finden.

Vorteile gegenüber Sylk (3.3)

Im Gegensatz zum Export in eine .slk-Datei bietet der Export in eine XML-Datei sowohl Farben, wie auch verschiedene Formattypen für die Zelle. So können dank Farben Tabellen übersichtlicher gestaltet werden und gewisse Formate wie Prozente oder Zahlen mit Kommastellen korrekt wiedergegeben werden. Ein weiterer netter Effekt des XML-Exports ist das erscheinende Icon von Excel wenn die richtige Process-Instruction im XML-File verwendet wird.

Einschränkungen gegenüber dem .xls/x Format

Leider unterstützt eine im XML-Format gespeicherte Exceldatei nicht alle Funktionalitäten, wie es sich der User von herkömmlichen Exceldateien (.xls, .xlsx) gewöhnt ist. So ist es zum Beispiel in einem im XML-Fomat gespeicherten Exceldokument nicht möglich Hierarchiestufen einzubauen, so dass definierte Knoten zusammenklappbar wären.

Eine weitere Einschränkung, die theoretisch zwar möglich wäre, sind die Zeileneinschübe. Ein Zeileneinschub würde durch die Zeichenfolge `
` in einem XMl-Text representiert werden. Das Problem hier ist das &-Zeichen, welches nicht in ein XML-Dokument geschrieben werden kann durch ein XDS-Script. Weil ein &-Zeichen eine besondere Bedeutung in XML hat, müsste das Zeichen durch die Zeichenfolge `&` ersetzt werden. Jedoch wird diese Zeichenfolge nicht interpretiert, wenn aus dem XDS-Script ein XML-Dokument generiert wird. Stattdessen, bleibt die Zeichenfolge erhalten und in der resultierenden XML-Datei steht an der Stelle wo der Zeileneinschub stehen sollte, representiert durch `
`, die Zeichenfolge `
`. Dies hat zur Folge, dass das erstellte Excel-Dokument im XML-Format keinen Zeileneinschub interpretiert, sondern nur den Text selber wiedergibt. Wenn zum Beispiel der Titel Reinigungsfläche aufgeteilt werden sollte in Reinigungs-fläche, dann würde als Ergebnis aus dem Script in Excel Reinigungs-
fläche stehen (ohne Zeileneinschub), weil die Zeichenfolge `&` jetzt als & interpretiert wurde, aber der Rest (`#10;`) keine Bedeutung hat.

Eine ebenfalls theoretisch mögliche, aber (bis jetzt) nicht umsetzbare Funktionalität ist das Einfrieren von Zeilen oder Spalten. Möglich ist es darum, weil wenn man eine .xlsx-Datei mit Daten füllt und beispielsweise die erste Zeile einfriert und das Dokument mithilfe von *Speichern unter* als XML-Kalkulationstabelle 2003 abspeichert, ist die erste Zeile, auch nach einem erneuten Öffnen, noch vorhanden. Im XML-Dokument selber, wenn man es mit einem Texteditor öffnet, ist erkennbar, dass am Ende des Dokuments im WorksheetOptions-Tag entsprechende Elemente hinzugefügt wurden, welche ein Fixieren der ersten Zeile definieren. Das seltsame daran ist, dass das WorksheetOptions-Tag nochmals den Excel-Namespace definiert, obwohl dieser jeweils schon eine Hierarchiestufe

weiter oben im Workbook-Tag in der namespace-Variable *x* definiert wurde. Wenn man also alles gleichermassen in den WorksheetOptions definieren würde, sollte das einfrieren auch funtionieren, wenn der namespace nicht nochmals definiert wird, aber vor jedem Element ein *x:* hinzugefügt wird, so dass das jeweilige Tag dem Excel-namespace zugeordnet wird. Leider ist dies jedoch nicht der Fall. Wenn man nun zum Testen, wieso es nicht funktioniert, manuell in Excel die erste Zeile fixiert und wiederum das XML-Dokument in einem Texteditor öffnet, erkennt man, dass Excel die zuvor erstellten Tags ignoriert und zuvor wiederum die eigene Definition (wie oben beschrieben) hineinfügt. So sind dann zwei WorksheetOptions-Tags vorhanden, wobei das aus dem XDS-Script generierte Tag und alle Unterknoten offensichtlich ignoriert werden. Den Excel-namespace nochmals neu zu definieren ist mit dem XDS-Script nicht möglich, weil dann alle Unterknoten, wenn kein *x:* am Anfang der Definition hinzugefügt wird, automatisch einen leeren Namespace definieren, was natürlich falsch ist.

Beispiele

Ein Beispiel für ein minimales Excel-XML-File ist hier zu finden und sieht folgendermassen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<?mso-application progid="Excel.Sheet"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">
  <Worksheet ss:Name="Sheet1">
    <Table>
      <Row>
        <Cell>
          <Data ss:Type="String">Employee No</Data>
        </Cell>
        <Cell>
          <Data ss:Type="String">Employee Name</Data>
        </Cell>
        <Cell>
          <Data ss:Type="String">Job</Data>
        </Cell>
      </Row>
      <Row>
        <Cell>
          <Data ss:Type="Number">7839</Data>
        </Cell>
        <Cell>
          <Data ss:Type="String">KING</Data>
        </Cell>
        <Cell>
          <Data ss:Type="String">PRESIDENT</Data>
        </Cell>
      </Row>
      <!-- More rows -->
    </Table>
  </Worksheet>
</Workbook>
```

Die Processinstruction `<?mso-application progid="Excel.Sheet"?>` sorgt dafür, dass das erstellte XML-File ein Excel-Icon erhält und so von jedem User direkt als Excel-Datei erkannt wird.

Bei der Anwendung mit einem XDS-Script ist es am Einfachsten, wenn kein default namespace verwendet wird. Stattdessen sollte man vor jedem Element ein „ss:" hinzufügen, so dass der korrekte namespace zur Spreadsheet Referenz verwendet wird.

Die Dokumentation des von Microsoft definierten XMLs, das als namespace einzubinden ist, beinhaltet alle verwendbaren Tags/Attribute und erklärt deren Verwendung.

Achtung: Zum Übertragen **mehrzeiliger Texte** müssen folgende Massnahmen ergriffen werden:

1. Ein mehrzeiliger Style muss definiert und verwendet werden.
2. Der mehrzeilige Text sollte entweder mit <![CDATA[…]]> geschrieben werden oder
 statt Zeilenumbrüche (linefeed) enthalten.

Beispiel für mehrzeilige Texte:

```
	<Styles>
		<Style ss:ID="mehrZeilig">
			<Alignment ss:Vertical="Top" ss:WrapText="1" />
		</Style>
	</Styles>
…
		<Cell ss:StyleID="mehrZeilig">
			<Data ss:Type="String">
				<![CDATA[erste Zeile
zweite Zeile]]>
			</Data>
		</Cell>
		<Cell ss:StyleID="mehrZeilig">
			<Data ss:Type="String">erste Zeile&#10;zweite Zeile</Data>
		</Cell>
```

Ein funktionierendes Beispiel für ein XDS-Script das eine Exceldatei im XML-Format generiert:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="fx" Value="=OSENV('temp') + '\Excel_Buildings.xml'" />
  </Variables>
  <Script>
    <WriteXML RootName="ss:Workbook" FileName="fx">
      <XMLattribute Name="'xmlns:x'"
                    Value="'urn:schemas-microsoft-com:office:excel'"/>
      <XMLattribute Name="'xmlns:ss'"
                    Value="'urn:schemas-microsoft-com:office:spreadsheet'"/>

      <XMLProcessingInstructions Target="'mso-application'"
                                 Data="'progid='+#34+'Excel.Sheet'+#34"/>

      <!-- Zellen-Formatdefinitionen -->
      <XMLelement Name="'ss:Styles'">

        <XMLelement Name="'ss:Style'">

          <XMLattribute Name="'ss:ID'" Value="'Default'"/>
          <XMLattribute Name="'ss:Name'" Value="'Normal'"/>
          <XMLelement Name="'ssAlignment'">
```

```
      <XMLattribute Name="'ss:Vertical'" Value="'Bottom'"/>
    </XMLelement>
    <XMLelement Name="'ss:Font'">
      <XMLattribute Name="'ss:FontName'" Value="'Calibri'"/>
      <XMLattribute Name="'x:Family'" Value="'Swiss'"/>
      <XMLattribute Name="'ss:Size'" Value="11"/>
      <XMLattribute Name="'ss:Color'" Value="'#000000'"/>
    </XMLelement>
  </XMLelement>
  <XMLelement Name="'ss:Style'">

    <XMLattribute Name="'ss:ID'" Value="'sGreen'"/>
    <XMLattribute Name="'ss:Name'" Value="'Green'"/>
    <XMLelement Name="'ss:Interior'">
      <XMLattribute Name="'ss:Color'" Value="'Green'"/>
      <XMLattribute Name="'ss:Pattern'" Value="'Solid'"/>
    </XMLelement>
  </XMLelement>
  <XMLelement Name="'ss:Style'">

    <XMLattribute Name="'ss:ID'" Value="'sBlue'"/>
    <XMLattribute Name="'ss:Name'" Value="'Blue'"/>
    <XMLelement Name="'ss:Interior'">
      <XMLattribute Name="'ss:Color'" Value="'Blue'"/>
      <XMLattribute Name="'ss:Pattern'" Value="'Solid'"/>
    </XMLelement>
  </XMLelement>
</XMLelement>

<!-- jetzt die Tabellendaten -->
<XMLelement Name="'ss:Worksheet'">
  <XMLattribute Name="'ss:Name'" Value="'Tabelle1'"/>

  <XMLelement Name="'ss:Table'">

    <Navigation>
      START INSTANCES Building
    </Navigation>

    <XMLelement Name="'ss:Row'">
      <ForEachObject>
        <IterateNav>
          VIA Object_To_Children
          CLASS bisB_OrganisationUnit
        </IterateNav>
        <!-- jetzt die Zellen der Zeile -->
        <XMLelement Name="'ss:Cell'">
          <SetVar Name="place" BisAttribute="bisB_Place"/>
          <SetVar Name="style" Value="Default"/>
          <!-- verschiedene Styles für Basel oder Luzern -->
          <IF Condition="place = 'Basel'">
            <SetVar Name="style" Value="sGreen"/>
          <ELSE Condition="place = 'Luzern'"/>
            <SetVar Name="style" Value="sBlue"/>
```

```
                </IF>
                <XMLattribute Name="'ss:StyleID'" Value="style"/>
                <XMLelement Name="'ss:Data'">
                  <XMLattribute Name="'ss:Type'" Value="'String'"/>
                  <SetVar Name="v" BisAttribute="Object_Display_Kontext"/>
                  <XMLtext Value="v"/>
                </XMLelement>
              </XMLelement>
            </ForEachObject>
          </XMLelement>
        </XMLelement>
      </XMLelement>
    </WriteXML>
  </Script>
</Defs>
```

3.5 Datenexport an externes Programm über XML Datei

Das Beispiel zeigt, wie eine XML Datei generiert wird. Die Ausgangsobjekte werden als Kontextobjekte an das Skript übergeben. Danach wird diese XML-Datei an ein Programm übergeben, welches eine ‚Resultat-Datei' generiert. Die Resultat-Datei wird nach einem Fehlereintrag durchsucht, um eine entsprechende Meldung zu zeigen. Der gesamte Prozess wird durch eine Fehlerbehandlung gekapselt.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="xmlfn" Value="=OSENV('TEMP')+ '\' + 'PQMessungen.xml'"/>
    <Var Name="PQlog" Value="=OSENV('TEMP')+ '\' + 'PQLogMess.txt'"/>
    <Var Name="PQProg" Value="=PARAM('HIP_PQPrgVerz') + 'Kurven.exe'"/>
    <!--
      FILECONTENTS ergibt den Datei-Inhalt. Dies wird hier nicht angewendet, vgl. *
    -->
    <Var Name="errtxt" Value="=FILECONTENTS(PQlog)"/>
  </Variables>
  <Script>
    <Progress Title="'PQ-Messungen exportieren'"/>
    <SetVar Name="i" Expression="0"/>
    <Progress TextOben="'XML-Datei erzeugen'"/>
    <FetchError BisUpdate="no">
      <!-- xml Datei für PQ-Programm generieren -->
      <WriteXML RootName="HIP" FileName="xmlfn">
        <SetVar Name="c" BisAttribute="COUNT"/>
        <ForEachObject>
          <SetVar Name="i" Expression="i+1"/>
          <IF Condition="i=1">
            <SetVar Name="tmpmsg"/>
            <Navigation>
              VIA HIP_MessungToStation
            </Navigation>
            <!-- beim ersten Objekt wird sich die Station gemerkt -->
            <SetVar Name="anStation"/>
            <GetVar Name="tmpmsg"/>
          </IF>
          <SetVar Name="vStaNum" BisAttribute="HIP_Stationsnummer">
            <Navigation>
              VIA HIP_MessungToStation
            </Navigation>
          </SetVar>
          <Progress TextUnten="'Messung Nr. ' + FORMAT(i, '%d')"
                    Position="(i / c) - 0.01" />
          <XMLelement Name="'Messung'">
            <XMLelement Name="'ID'" BisAttribute="HIP_PQPunktID" />
            <XMLelement Name="'Action'" Value="'Save'" />
            <XMLelement Name="'EDVNr'" Value="vStaNum" />
            <XMLelement Name="'WMNr'" BisAttribute="HIP_WM_Nummer" />
            <XMLelement Name="'Status'" BisAttribute="HIP_WM_Status"
                        Format="%d" />
```

```
          <XMLelement Name="'Messdatum'" BisAttribute="HIP_Messbeginn"
                      Format="yyyy-mm-dd" />
          <XMLelement Name="'Autor'">
            <XMLelement Name="'Vorname'" BisAttribute="M2P%Vorname" />
            <XMLelement Name="'Nachname'" BisAttribute="M2P%Nachnm" />
          </XMLelement>
          <XMLelement Name="'P'" BisAttribute="HIP_Pegel" />
          <XMLelement Name="'Q'" BisAttribute="HIP_Abfluss" />
          <XMLelement Name="'Fluegeldaten'">
            <SetVar Name="tmpmsg"/>
            <Navigation>
              VIA Object_To_Children
              TYPE HIP_Fluegeldaten
              SORT (+HIP_FluegelNr)
            </Navigation>
            <ForEachObject>
              <XMLelement Name="'Nr'" BisAttribute="HIP_FluegelNr"/>
              <XMLelement Name="'FluegelNr'" BisAttribute="D2E%FNr"/>
            </ForEachObject>
            <GetVar Name="tmpmsg"/>
          </XMLelement>
          <XMLelement Name="'F'" BisAttribute="HIP_Flaecheninhalt"/>
        </XMLelement>
      </ForEachObject>
    </WriteXML>
    <LogEntry Value="FORMAT(i, '%d Messungen in ') + xmlfn"/>
    <!-- Log file von PQ-Programm löschen -->
    <File Name="PQlog" Do="delete"/>
    <!-- Messungen an PQ-Programm übergeben -->
    <Progress TextOben="'Messungen an PQ-Programm übergeben'" TextUnten="''"/>
    <!--
      Programm wird ausgeführt, durch ‚wait' wird die Beendigung des Programms
      abgewartet, damit die ‚PQlog' Datei ausgewertet werden kann.
    -->
    <ShellExec File="PQProg" Wait="yes">' /import:Msg /source:"'
               + xmlfn + '" /log:"' + PQlog + '"'
    </ShellExec>
    <!-- Log file von PQ-Programm lesen -->
    <Progress TextOben="'Log-Datei vom PQ-Programm lesen'"/>
    <SetVar Name="logerr" Expression="0"/>
    <!--
      * Die ‚PQlog' Datei wird gelesen. Jede Zeile wird auf ‚E:' getestet.
      Alternativ könnte dieser Text in der Variable 1) gesucht werden (STRPOS)
    -->
    <OpenFile Mode="read" FileName="PQlog" RecVar="logrec">
      <LogEntry Value="'Record ' + logrec"/>
      <IF Condition="SUBSTR(logrec,1,2) = 'E:'">
        <SetVar Name="logerr" Expression="logerr + 1"/>
      </IF>
    </OpenFile>
  <OnError/>
    <!-- Die Zeile wird nur im Fehlerfall erreicht -->
    <ShowMessage Msg="'Fehler ' + varLastError" Type="error"/>
    <LogEntry Type="error" Value="'Skript: ' + varLastError"/>
```

```
    </FetchError>
    <!-- Log file von PQ-Programm in BIS speichern -->
    <!-- Meldung anzeigen (OK oder Fehler) -->
    <IF Condition="logerr &gt; 0">
      <ShowMessage Msg="FORMAT(logerr, '%d Fehler im PQ aufgetreten!')
             + #13 + #10 + 'Soll die Protokolldatei angezeigt werden?'"
                   Type="warning" Buttons="yes,no" ResultVar="rv"/>
      <IF Condition="rv='6'">
        <ShellExec File="PQlog" Wait="no"/>
      </IF>
    <ELSE/>
      <ShowMessage Type="information"
                   Msg="FORMAT(i, '%d Messung(en) exportiert')"/>
    </IF>
  </Script>
</Defs>
```

3.6 Datenexport in eine XML Datei

Das Beispiel generiert dieselbe XML Datei, wie die Konfiguration XDS_XMLRaumExport (vgl. Demo Datenbank, Raumbuch: Menu Extra „BISXDS-XML-Export" „Gebäude in XML Datei".

Derselbe Export wird im nächsten Beispiel mit Dialog Elementen angereichert.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs Version="20">
  <Variables>
    <Var Name="fn" Value="=OSENV('ALLUSERSPROFILE') +
'\ByronBIS\DemoDb\XDS\BuildingRooms2.xml'"/>
  </Variables>

  <Script>
    <Navigation>
      CLASS Building
    </Navigation>
    <WriteXML RootName="BuildingDocument" FileName="fn">
      <ForEachObject>
        <XMLelement Name="'Gebaeude'">
          <SetVar Name="sid" BisAttribute="bisB_SpaceId"/>
          <XMLattribute Name="'Nr'" Value="sid"/>
          <Navigation>
            VIA Object_To_Children
            CLASS Floor
          </Navigation>
          <ForEachObject>
            <XMLelement Name="'Geschoss'">
              <SetVar Name="sid" BisAttribute="bisB_SpaceId"/>
              <XMLattribute Name="'Nr'" Value="sid"/>
              <XMLelement Name="'Raeume'">
                <Navigation>
                  VIA Object_To_Children
                  CLASS Room
                </Navigation>
                <ForEachObject>
                  <XMLelement Name="'Raum'">
                    <SetVar Name="sid" BisAttribute="bisB_SpaceId"/>
                    <XMLattribute Name="'Nr'" Value="sid"/>
                    <SetVar Name="sid" BisAttrib-
ute="bisC_RoomToFlooring%Object_Display_Kontext"/>
                    <XMLattribute Name="'Bodenbelag'" Value="sid"/>
                    <XMLelement Name="'DIN277'">
                      <SetVar Name="sid" BisAttrib-
ute="bisC_SpaceToSpaceUsageType%bisC_SpaceUsageNumber"/>
                      <XMLattribute Name="'Nr'" Value="sid"/>
                    </XMLelement>
                    <XMLelement Name="'KST'">
                      <SetVar Name="sid" BisAttrib-
ute="bisC_ObjectToCostCenter%Object_Display_Kontext"/>
                      <XMLtext Value="sid"/>
                    </XMLelement>
                  </XMLelement>
```

```
              </ForEachObject>
            </XMLelement>
          </XMLelement>
        </ForEachObject>
      </XMLelement>
    </ForEachObject>
  </WriteXML>
  <ShellExec File="fn" Wait="no"/>
 </Script>
</Defs>
```

3.7 Datenexport in eine XML Datei mit Dialogelemente

Der Export ist derselbe wie im vorangegangenen Beispiel. Es werden zusätzlich Möglichkeiten von Dialogelementen gezeigt. Diese machen eine explizite Transaktionssteuerung notwendig. Damit die Fortschrittsanzeige auch wahrgenommen werden kann wurde eine WHILE Schleife als ‚Bremse' eingebaut.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs Version="20">
  <Variables>
    <Var Name="fn"
         Value="=OSENV('ALLUSERSPROFILE') +
'\ByronBIS\DemoDb\XDS\BuildingRooms2.xml'" />
  </Variables>

  <Script>
    <Progress Title="'Gebäude in XML exportieren'"
              TextOben="'Datei schreiben'" Position="0.01" />
    <FetchError BisUpdate="No">
      <SetVar Name="c"/>
      <Navigation>
        LOOP (VIA Object_To_Children)
        CLASS Room
      </Navigation>
      <SetVar Name="total" BisAttribute="COUNT"/>
      <GetVar Name="c"/>
      <SetVar Name="c" Expression="0"/>
      <Navigation>
        CLASS Building
      </Navigation>
      <WriteXML RootName="BuildingDocument" FileName="fn">
        <ForEachObject>
          <XMLelement Name="'Gebaeude'">
            <SetVar Name="sid" BisAttribute="bisB_SpaceId"/>
            <Progress TextOben="sid"/>
            <XMLattribute Name="'Nr'" Value="sid"/>
            <Navigation>
              VIA Object_To_Children
              CLASS Floor
            </Navigation>
            <ForEachObject>
              <XMLelement Name="'Geschoss'">
                <SetVar Name="sid" BisAttribute="bisB_SpaceId"/>
                <Progress TextUnten="sid"/>
                <XMLattribute Name="'Nr'" Value="sid"/>
                <XMLelement Name="'Raeume'">
                  <Navigation>
                    VIA Object_To_Children
                    CLASS Room
                  </Navigation>
                  <ForEachObject>
                    <XMLelement Name="'Raum'">
                      <SetVar Name="c" Expression="c + 1"/>
                      <Progress Position="0.9 * (c / total)"/>
```

```
                    <SetVar Name="sid" BisAttribute="bisB_SpaceId"/>
                    <XMLattribute Name="'Nr'" Value="sid"/>
                    <SetVar Name="sid" BisAttrib-
ute="bisC_RoomToFlooring%Object_Display_Kontext"/>
                    <XMLattribute Name="'Bodenbelag'" Value="sid"/>
                    <XMLelement Name="'DIN277'">
                      <SetVar Name="sid" BisAttrib-
ute="bisC_SpaceToSpaceUsageType%bisC_SpaceUsageNumber"/>
                      <XMLattribute Name="'Nr'" Value="sid"/>
                    </XMLelement>
                    <XMLelement Name="'KST'">
                      <SetVar Name="sid" BisAttrib-
ute="bisC_ObjectToCostCenter%Object_Display_Kontext"/>
                      <XMLtext Value="sid"/>
                    </XMLelement>
                    <!-- Bremse -->
                    <SetVar Name="j" Expression="0"/>
                    <WHILE Condition="j &lt; 1000">
                      <SetVar Name="j" Expression="j + 1"/>
                    </WHILE>
                  </XMLelement>
                </ForEachObject>
              </XMLelement>
            </XMLelement>
          </ForEachObject>
        </XMLelement>
      </ForEachObject>
    </WriteXML>
  </FetchError>
  <!--Progress Position="1.0"/-->
  <ShowMessage Msg="'Datei anzeigen?'"
               Type="confirmation"
               Buttons="yes,no" ResultVar="rv"/>
  <IF Condition="rv='6'">
    <ShellExec File="fn" Wait="no"/>
  </IF>
 </Script>
</Defs>
```

3.8 Positionsobjekte in CAD Plan exportieren

Das Beispiel geht von folgendem Modell aus: Es gibt Vorlagepläne, diese sind von der Klasse BSP_Plan und die Eigenschaft BSP_PlanTyp hat den Wert 30. Diese Pläne sind über die Assoziation BSPaPlanToPxt mit dem Zielplan verknüpft. Die Zieldatei ist also als Objekt bereits angelegt, die Datei (Attribut Doc_Path) nicht unbedingt.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Script>
    <Navigation>
      START INSTANCES BSP_Plan
      VALUE BSP_PlanTyp = 30
      VIA BSPaPlanToPxt
      [
         COMPARE bisB_ModificationDate () &lt; (VIA BSPaPxtToPlan)
      OR
        VALUE bisB_FileExists = false
      ]
    </Navigation>
    <ForEachObject>
      <CadExport Layer="E_BIS_Pxt"
                 FontName="Verdana"
                 FontSize="40">
        <TargetFile/>
        <Navigation>
          VIA BSPaPxtToPlan
        </Navigation>
        <SourceFile/>
        <Navigation>
          VIA Doc_Referenced_By
        </Navigation>
        <UseTransformation/>
        <Navigation>
           [
           OR
             LOOP (VIA Object_To_Children)
           ]
           VIA Room_To_Component
           CLASS BSPaFlaechenverbinder
        </Navigation>
        <ForEachObject>
          <SetVar Name="key" Attribute="BSPaPxtKey"/>
          <TextGraphic Value="key"/>
        </ForEachObject>
      </CadExport>
    </ForEachObject>
  </Script>
</Defs>
```

3.9 HTML E-Mail versenden mit eingebettetem Attachment / Bild

Aus Sicherheitsgründen wird oft konfiguriert, dass Bilder, sofern Sie von "extern" verlinkt werden, in einer E-Mail entfernt werden. Um dies zu verhindern, muss z.B. ein Logo als E-Mail Anhang (Attach-

ment) mitgeliefert werden und in der Nachricht - im <img>-Tag - referenziert werden. Dies kann über die "cid" (Content-ID) und den Dateinamen gemacht werden:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Script>
    <MailSend Host="mail.byron.ch"
              From="'frei@byron.ch'"
              To="'frei@byron.ch'"
              ContentType="multipart/mixed; charset=iso-8859-1"
              Subject="'Tescht'">
      <Text Contents="'&lt;img src=&quot;cid:ByronIcon.png&quot;/> &lt;br/>&lt;br/>
Mein Text-Inhalt'"
            ContentType="text/html; charset=iso-8859-1"/>

      <Attachment Name="'C:\Users\frei\Desktop\ByronIcon.png'"
                  ContentDisposition="attachment"
                  ContentType="image/png"/>
    </MailSend>
  </Script>
</Defs>
```

3.10 HTML E-Mail versenden

Das Beispiel zeigt die Anwendung von , zum versenden einer E-Mail. Dazu wird die relevante Information zuerst in eine XML Datei geschrieben. Nachher wird jede <Person> in dieser XML Datei mit einem Stylesheet transformiert. Im 1. Beispiel wird das Resultat der Stylesheet Transformation in eine Variable geschrieben und versendet. Im 2. Beispiel wird die Transformation in eine Datei geschrieben.

In beiden Fällen wird zuerst die gesamte Information in eine XML Datei geschrieben. Dies könnte man auch anders machen, z.B. Indem man pro E-Mail eine XML Datei generiert.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="fx" Value="=OSENV('Temp') + '\_auftraege.xml'" />
  </Variables>

  <Script>
    <!-- relevante Daten suchen -->
    <Navigation>
      START INSTANCES bisP_MaintOrder
      VALUE bisP_OrderState &lt; 8
      COUNT (
        VIA bisP_MaintOrderToMaintPerson VALUE bisB_Email '*'
      ) > 0
    </Navigation>
    <SetVar Name="cnt" BisAttribute="COUNT"/>
    <LogEntry Value="'Aufträge: ' + FORMAT(cnt, ' %d') + ' in Datei ' + fx" />
    <!--
      Daten in eine XML Datei schreiben FormattedOutput ist
      abgeschaltet, da mit Formatierung die Datei ohne BOM
      geschrieben wird und damit als UTF-8 interpretiert wird
    -->
```

```
    <WriteXML RootName="AlleJobs" FileName="fx" FormattedOutput="no">
      <ForEachObject>
        <BuildGroup GroupBy="durchfuehrer">
          <SetVar Name="durchfuehrer"
                  BisAttribute="bisP_MaintOrderToMaintPerson%bisB_ObjectImage" />
        </BuildGroup >
        <XMLelement Name="'Person'">
          <XMLelement Name="'Name'">
            <SetVar Name="v" BisAttrib-
ute="bisP_MaintOrderToMaintPerson%Object_Display_Kontext" />
            <XMLtext Value="v" />
          </XMLelement>
          <XMLelement Name="'Mail'">
            <SetVar Name="v" BisAttribute="bisP_MaintOrderToMaintPerson%bisB_Email"
/>
            <XMLtext Value="v" />
          </XMLelement>
          <ForEachObject>
            <XMLelement Name="'Auftrag'">
              <XMLelement Name="'Id'">
                <SetVar Name="v" BisAttribute="bisP_OrderNo" />
                <XMLtext Value="v" />
              </XMLelement>
              <XMLelement Name="'CreationDate'">
                <SetVar Name="v" Format="dd.mm.yyyy"
                        BisAttribute="bisB_CreationDate" />
                <XMLtext Value="v" />
              </XMLelement>
              <XMLelement Name="'Name'">
                <SetVar Name="v" BisAttribute="Object_Display_Kontext" />
                <XMLtext Value="v" />
              </XMLelement>
              <XMLelement Name="'bisP_Prepared'">
                <SetVar Name="v" BisAttribute="bisP_Prepared" />
                <XMLtext Value="v" />
              </XMLelement>
              <XMLelement Name="'bisP_ObjectBasedResponses'">
                <SetVar Name="v" BisAttribute="bisP_ObjectBasedResponses" />
                <XMLtext Value="v" />
              </XMLelement>
              <XMLelement Name="'Durchfuehrung'">
                <SetVar Name="v" BisAttrib-
ute="bisP_MaintOrderToMaintPerson%Object_Display_Kontext" />
                <XMLtext Value="v" />
              </XMLelement>
              <XMLelement Name="'ObjectImage'">
                <SetVar Name="v" BisAttribute="bisB_ObjectImage"/>
                <XMLtext Value="v"/>
              </XMLelement>
            </XMLelement>
          </ForEachObject>
        </XMLelement>
      </ForEachObject>
    </WriteXML>
```

```
    <!-- geschriebene XML Datei wieder lesen -->
    <ReadXML FileName="fx">
      <ForEachElement SelectNode="'Person'">
      <!--
        pro Person eine HTML durch xslt Transformation erzeugen.
        Der resultierende html Text befindet sich nachher in der
        Variable ‚MailInhalt'
      -->
      <TransformXSL ToVar="MailInhalt">

<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:output method="html"
        encoding="iso-8859-1"
        indent="yes"/>
  <xsl:template match="Person">
    <html>
      <head>
        <META http-equiv="Content-Type"
           content="text/html; charset=iso-8859-1"/>
        <title>Aufträge</title>
        <style type="text/css">
          .Text { font-size: 10pt; font-family: "Courier New";}
        </style>
      </head>
      <Body class="Text"
         bgcolor="#fafdfe">
        <style type="text/css">
          .Text { font-size:10pt; font-family:"Courier New"}
        </style>
        <table class="Text"
            bgcolor="#e4f0fc"
            width="682"
            border="0"
            cellspacing="0"
            cellpadding="0">
          <tr>
            <td width="5"
              rowspan="4">
              <img border="0"
                 src="http://www.byron.ch/images/spacer.gif"/>
            </td>
            <td width="12">
              <img border="0"
                 src="http://www.byron.ch/images/spacer.gif"/>
            </td>
            <td width="128"
              align="left"
              class="LogoCell" >
              <img src="http://www.byron.ch/images/byron_logo.gif"
                 alt="Logo der Byron Informatik AG"/>
            </td>
            <td width="537"
```

```
          colspan="2">
          <img src="http://www.byron.ch/images/header_text.jpg"/>
        </td>
      </tr>
    </table>
    <br/>
    <br/>
    <b>
      Guten Tag <xsl:value-of select="Name"/>
    </b>
    <br/>
    <br/>
    <b>Folgende Aufträge wurden neu erfasst:</b>
    <br/>
    <br/>
    <table class="Text"
        border="1"
        width="100%">
      <tr bgcolor="#99CCFF">
        <td valign="top">
          Erfasst
        </td>
        <td valign="top">
          Auftrags ID
        </td>
        <td valign="top">
          Bezeichnung
        </td>
        <td valign="top">
          Objekt basierend
        </td>
        <td valign="top">
          Durchführung
        </td>
      </tr>

      <xsl:for-each select="Auftrag">
        <tr>
          <td>
            <xsl:value-of select="CreationDate"/>
          </td>
          <td>
            <A>
              <xsl:attribute name="Href">

http://www.byron.ch/supportdatenbank/goto/bisreleaseinfo.asp?AuftragsID=<xsl:value-
of select="Id"/>
              </xsl:attribute>
              <xsl:value-of select="Id"/>
            </A>
          </td>
          <td>
            <xsl:value-of select="Name"/>g
          </td>
```

```
              <td>
                <xsl:choose>
                  <xsl:when test ="bisP_ObjectBasedResponses='true'">Ja</xsl:when>
                  <xsl:otherwise>Nein</xsl:otherwise>
                </xsl:choose>
              </td>
              <td>
                <xsl:value-of select="Durchfuehrung"/>
              </td>
            </tr>

          </xsl:for-each>
        </table>
        <br/>
        <br/>
        Byron Informatik AG
                  
        Tel:
             
        +41 61 690 96 00   
        <br/>
        Efringerstr. 32

              
;
        Fax:
             
        +41 61 690 96 09   
        <br/>
        CH-4057 Basel

              
;  
        E-mail:
          
        <A href="mailto:support@byron.ch">support@byron.ch</A>
           
        <br/>

              
; 

              
;
        Internet: 
        <A href="http://www.byron.ch/">http://www.byron.ch</A>
        <br/>
        <br/>
        Diese Mail wurde automatisch generiert durch <A
href="http://www.byron.ch/Produkte/bis/BisXds.htm">Byron/BIS XDS</A>.
      </Body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

```
      </TransformXSL>
      <!-- Info für Betreff aus XML Datei holen -->
      <NodeToVar VarName="vname" SelectNode="'Name'" />
      <!-- Empfänger Adresse aus XML Datei holen -->
      <NodeToVar VarName="vToAdr" SelectNode="'Mail'" />
      <LogEntry Value="'person mail: ' + vToAdr " />
      <!-- ObjectImage von Objekt aus XML Datei holen -->
      <NodeToVar VarName="vObjectImage" SelectNode="'ObjectImage'" />

      <!--
        E-Mail senden. Inhalt wurde von TransformXSL in MailInhalt
        geschrieben
      -->
      <FetchError BisUpdate="yes">
        <MailSend Host="mail.byron.ch"
                  From="'support@byron.ch'"
                  To="vToAdr"
                  ContentType="text/html; charset=iso-8859-1"
                  Subject="'Auftrag ' + vname"
                  Contents="MailInhalt"/>
        <!--
          Fehlerprüfung: Wenn kein Fehler aufgetreten,
          Filternavigation (Status setzen) ausführen.
        -->
        <Navigation>
          START DATAOF vObjectImage
          MODIFY "bisP_MailSend" "1"
        </Navigation>
        <OnError Abort="no"/>
          <!--
            Hier kommen wir im Fehlerfall
            hier könnte auch eine Mail an den Admin verschickt werden
          -->
          <LogEntry Value="'Fehler beim Mailsend aufgetreten: ' + varLastError"/>
        </FetchError>
      </ForEachElement>
    </ReadXML>
  </Script>
</Defs>
```

Das folgende 2. Beispiel schreibt die E-Mail in eine Datei. Die fehlenden Teile sind mit vorhergehendem Beispiel identisch. Die Abweichungen sind rot markiert.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="fn" Value="=OSENV('Temp') + '\_auftraege'" />
    <Var Name="fx" Value="=OSENV('Temp') + '\_auftraege.xml'" />
  </Variables>
  <Script>
    <!-- relevante Daten suchen und in XML Datei schreiben -->
    <WriteXML RootName="AlleJobs" FileName="fx" FormattedOutput="no">
    <!-- XML Daten wieder lesen -->
    <ReadXML FileName="fx">
      <ForEachElement SelectNode="'Person'">
        <TransformXSL Output="fn">
          <xsl:stylesheet version="1.0" ….
        </TransformXSL>
        <NodeToVar VarName="vname" SelectNode="'Name'" />
        <NodeToVar VarName="vToAdr" SelectNode="'Mail'" />
        <!-- MailSend ohne Contents, der kommt durch Attachment inline -->
        <MailSend Host="mail.byron.ch"
                  From="'support@byron.ch'"
                  To="vToAdr"
                  ContentType="text/html; charset=iso-8859-1"
                  Subject="'Auftrag ' + vname">
          <Attachment Name="fn"
                      ContentDisposition="inline"
                      ContentType="text/html" />
        </MailSend>
      </ForEachElement>
    </ReadXML>
  </Script>
</Defs>
```

Varianten eine Html Mail mit Anhang zu senden (nur mit den in diesem Zusammenhang wesentlichen Attributen):

```
<MailSend ContentType="multipart/mixed">
  <Text Contents="mailHtmlVar" ContentType="text/html" />
  <Attachment Name="grafikPDF"
              ContentType="application/pdf"
              ContentDisposition="attachment" />
  <Attachment Name="excelRepFile"
              ContentType="application/msexcel"
              ContentDisposition="attachment" />
</MailSend>
```

Wenn die Mail in eine Datei erzeugt wurde (hier mailHtmlFile)

```
<Text Name="mailHtmlFile" ContentType="text/html" />
```

Das kann man auch durch ein Attachment inline erreichen:

```
<MailSend ContentType="multipart/mixed">
  <Attachment Contents="mailHtmlFile"
```

```
              ContentType="text/html"
              ContentDisposition="inline" />
  <Attachment Name="grafikPDF"
              ContentType="application/pdf"
              ContentDisposition="attachment" />
  <Attachment Name="excelRepFile"
              ContentType="application/msexcel"
              ContentDisposition="attachment" />
</MailSend>
```

3.11 XML mit dynamischer Tiefe schreiben

Das Beispiel zeigt die Anwendung von <IterateNav>.

```
<Script>
  <WriteXML RootName="Organisationen" FileName="'C:\Test.xml'">
    <Navigation>
      START CLASS bisB_OrganisationContainerRoot
      VIA Object_To_Children
      CLASS bisB_Organisation
    </Navigation>
    <ForEachObject>
    <XMLelement Name="'Organisation'">
      <IterateNav>
        VIA Object_To_Children
        CLASS BisB_OrganisationUnit
      </IterateNav>
      <XMLattribute Name="'name'"
                    BisAttribute="Object_Display_Kontext" />
      </XMLelement>
    </ForEachObject>
  </WriteXML>
</Script>
```

Ergebnis:

```
<Organisationen>
  <Organisation name="A">
    <Organisation name="A1">
      <Organisation name="A1.1" />
    </Organisation>
    <Organisation name="A2" />
  </Organisation>
  <Organisation name="B">
    <Organisation name="B1" />
  </Organisation>
</Organisationen>
```

3.12 Termin transformieren

Das Beispiel zeigt wie man einen Termin in eine ‚beliebige' XML Struktur transformieren kann. In der Methodensprache liest man das Terminattribut und konvertiert es in den XML Text.

```
method return text is
  ex: boolean;
  val: termin;
begin
  Get_TerminEx(Search_View('My_DueDate'), self, ex, val);
  if ex then
    return TerminXMLTextOf(val);
  end if;
end;
```

Diesen Text lässt sich dan durch xsl in eine andere XML-Struktur transformieren.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="tmpxml">= OSENV('temp') + '\' +
      FORMAT(NOW, 'yyyymmddhhnnsszzz') + '.xml'
    </Var>
  </Variables>

  <Script>
    <SetVar Name="duedatexml" BisAttribute="My_DueDateXML" />
    <OpenFile Mode="create" FileName="tmpxml">
      <Next Variable="duedatexml" />
    </OpenFile>
    <ReadXML FileName="tmpxml">
      <TransformXSL Output ="OSENV('temp') + '\termintransformed.xml'">

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" omit-xml-declaration="yes"
              encoding="iso-8859-1" indent="yes" />
  <xsl:template match="/">
    <xsl:element name="CalendarItem">
      <xsl:apply-templates select="SimpleDate" />
      <xsl:apply-templates select="RepeatedDate" />
    </xsl:element>
  </xsl:template>
  <xsl:template match="Start">
    <xsl:element name="StartTime">
      <xsl:apply-templates select="Date"/>T
      <xsl:apply-templates select="Time"/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="End">
    <xsl:element name="EndTime">
      <xsl:apply-templates select="Date"/>T
      <xsl:apply-templates select="Time"/>
    </xsl:element>
  </xsl:template>
```

```
<xsl:template match="Date">
  <xsl:value-of select="." />
</xsl:template>
<xsl:template match="Time">
  <xsl:value-of select="." />
</xsl:template>
<xsl:template match="SimpleDate">
  <xsl:apply-templates select="Start"/>
  <xsl:apply-templates select="End"/>
  <xsl:element name="Repeat">
    <xsl:attribute name="cycle">once</xsl:attribute>
  </xsl:element>
</xsl:template>
<xsl:template match="Month">
  <xsl:element name="Months">
    <xsl:value-of select="name(node())" />
  </xsl:element>
</xsl:template>
<xsl:template match="DayInMonth">
  <xsl:element name="Days">
    <xsl:value-of select="DayInMonth" />
  </xsl:element>
</xsl:template>
<xsl:template match="Weekdays">
  <xsl:element name="Weekdays">
    <xsl:value-of select="concat(../WeekdayPos/., ' ', name(node()))"/>
  </xsl:element>
</xsl:template>
<xsl:template match="Yearly">
  <xsl:element name="Repeat">
    <xsl:choose>
      <xsl:when test="Repetition &gt; 1">
        <xsl:element name="Years">
          <xsl:value-of select="Repetition" />
        </xsl:element>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="cycle">yearly</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates select="Month" />
    <xsl:apply-templates select="DayInMonth" />
    <xsl:apply-templates select="Weekdays" />
  </xsl:element>
</xsl:template>
<xsl:template match="Monthly">
  <xsl:element name="Repeat">
    <xsl:choose>
      <xsl:when test="Repetition &gt; 1">
        <xsl:element name="Months">
          <xsl:value-of select="Repetition" />
        </xsl:element>
      </xsl:when>
      <xsl:otherwise>
```

```
          <xsl:attribute name="cycle">monthly</xsl:attribute>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:element>
  </xsl:template>
  <xsl:template match="Weekly">
    <xsl:element name="Repeat">
      <xsl:choose>
        <xsl:when test="Repetition &gt; 1">
          <xsl:element name="Days">
            <xsl:value-of select="Repetition * 7" />
          </xsl:element>
        </xsl:when>
        <xsl:otherwise>
          <xsl:attribute name="cycle">weekly</xsl:attribute>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:element>
  </xsl:template>
  <xsl:template match="Daily">
    <xsl:element name="Repeat">
      <xsl:choose>
        <xsl:when test="Repetition &gt; 1">
          <xsl:element name="Days">
            <xsl:value-of select="Repetition" />
          </xsl:element>
        </xsl:when>
        <xsl:otherwise>
          <xsl:attribute name="cycle">daily</xsl:attribute>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:element>
  </xsl:template>
  <xsl:template match="RepeatedDate">
    <xsl:apply-templates select="Start"/>
    <xsl:apply-templates select="End"/>
    <xsl:apply-templates select="Yearly" />
    <xsl:apply-templates select="Monthly" />
    <xsl:apply-templates select="Weekly" />
    <xsl:apply-templates select="Daily" />
  </xsl:template>
</xsl:stylesheet>

      </TransformXSL>
    </ReadXML>
  </Script>
</Defs>
```

3.13 Personen Import mit LDAP (ADO)

Das Beispiel zeigt wie Personen abgeglichen werden. Der Schlüssel ist die mail Adresse. Der Zugriff auf die Personendaten erfolgt über das LDAP durch eine ADO Implementierung (Active Directory Provider). Im Select werden nur records mit Nachname (sn) und Mailadresse (mail) abgefragt.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs>
  <Variables>
    <Var Name="cns"
         Value="='Provider=ADsDSOObject; User ID=Username; Password=pst'" />
    <Var Name="sel">=
      'select mail, samaccountname, sn, givenName, name from ' +
      ' ''LDAP://company.com'' where sn=''*'' and mail =''*'' '
    </Var>
    <Var Name="vLoginName" Value="='' + vAccountName" />
  </Variables>

  <Script>
    <LogEntry Value="'byron LDAP'"/>
    <ADOconnection Id="ado" ConnectionString="cns" Select="sel">
      <ADOconnectionParameter Name="Page size" Value="20000" />
      <ADOfirst Id="ado" Variable="ok" />
      <SetVar Name="Anzahl" Expression="0" />
      <WHILE Condition="ok">
        <ADOgetField Id="ado" GetByName="'mail'"
                     Variable="vMail" />
        <ADOgetField Id="ado" GetByName="'samaccountname'"
                     Variable="vAccountName" />
        <ADOgetField Id="ado" GetByName="'sn'"
                     Variable="vLastname" />
        <ADOgetField Id="ado" GetByName="'givenName'"
                     Variable="vFirstname"/>
        <ADOgetField Id="ado" GetByName="'name'"
                     Variable="vFullname" />
        <Navigation>
          START VALUE bisB_EMail = DATAOF vMail
          [
          ELSIF = 0
            CREATE 1 Person ()
            MODIFY bisB_EMail DATAOF vMail
          ]
          MODIFY bisB_LoginName DATAOF vAccountName
          MODIFY bisB_Firstname DATAOF vFirstname
          MODIFY bisB_Lastname DATAOF vLastname
        </Navigation>
        <LogEntry Value="'Email: ' + vMail" />
        <SetVar Name="Anzahl" Expression="Anzahl + 1" />
        <ADOnext Id="ado" Variable="ok" />
      </WHILE>
    </ADOconnection>
    <LogEntry Value="'Anzahl: ' + FORMAT(Anzahl, '%d')" />
  </Script>
</Defs>
```

3.14 Kalendertermine synchronisieren

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Defs Version="20">

  <Script>
    <SetVar Name="nl" Expression="#13+#10"/>
    <Logfile DirectoryName="'C:\Temp'"
        FileName="'log'"
        ExtensionName="'txt'"
        Level="information"/>

    <!--
    Ablauf:
      1. Alle doppelt geänderten ausfindig machen, Kombinationen von
         gelöscht-geändert berücksichtigen.
      2. Alle auf dem Server gelöschten lokal löschen.
      3. Alle lokal geänderten und neuen an den Server schicken.
      4. Alle lokal gelöschten vom Server löschen.
      5. Alle auf dem Server geänderten oder neuen lokal speichern.
    -->

    <!--
      Stati für dieses Beispiel:
        0=neu
        1=geändert
        2=synchron mit Server
        3=Synchronisationsproblem
        4=als gelöscht markiert
    -->

    <FetchError>
      <CalDavConnection Host="'https://mail.byron.ch'"
          CalendarLocation="'/calendars/byron.ch/knecht/Calendar/'"
          Username="'knecht'"
          Password="'********'">

        <!-- Als Erstes werden alle Server-Events, an denen man
interessiert ist in einer Lookup-Struktur gespiegelt -->
        <FetchError>
          <CalDavForEachServerEvent EtagVar="etag"
              FileNameVar="filename"
              SearchFilter="
              '&lt;c:comp-filter name=&quot;VCALENDAR&quot;&gt; '+
                '&lt;c:comp-filter name=&quot;VEVENT&quot;&gt; ' +
                  '&lt;c:time-range start=&quot;20090101T000000Z&quot; ' +
                      'end=&quot;20160101T000000Z&quot;/&gt; ' +
                '&lt;/c:comp-filter&gt; ' +
              '&lt;/c:comp-filter&gt;'">
            <AddLookup Name="ServerEvents" KeyValue="filename"
StoreValue="etag"/>
          </CalDavForEachServerEvent>
        <OnError/>
          <LogEntry Value="'Konnte Eventliste nicht vom Server lesen' +
```

```
            varLastError" Type="error"/>
        </FetchError>

        <FetchError BisUpdate="no">
          <Navigation>
            START INSTANCES bisB_CalDavEvent
          </Navigation>
        </FetchError>
        <ForEachObject>
          <SetVar Name="dbFilename" BisAttribute="bisB_CalDavFilename"/>
          <SetVar Name="dbEtag" BisAttribute="bisB_CalDavEtag"/>
          <SetVar Name="status" Format="AsInteger"
              BisAttribute="bisB_CalDavStatus"/>
          <SetVar Name="surr" BisAttribute="bisB_ObjectImage"/>

          <GetLookup Name="ServerEvents" KeyValue="dbFilename"
ToVar="etag"/>

          <!-- Falls Objekt lokal geändert, prüfen, ob es auf dem Server auch geän-
dert oder gelöscht wurde, falls ja => Synchronisationsproblem -->
          <IF Condition="(status = 1) AND ((etag = NULL) OR (etag &lt;&gt;
dbEtag))">
            <FetchError BisUpdate="yes">
              <Navigation>
                MODIFY bisB_CalDavStatus 3 -- Synchronisationsproblem
              </Navigation>
            <OnError/>
              <LogEntry Value="'Konnte Event nicht ändern' + varLastError"
                  Type="error"/>
            </FetchError>

          <!-- Falls lokal nicht geändert und auf dem Server fehlend, lokal auch
löschen -->
          <ELSE Condition="(status = 2) AND (etag = NULL)"/>
            <FetchError BisUpdate="yes">
              <Navigation>
                START VALUE bisB_CalDavFileName = DATAOF dbFilename
                DELETE
              </Navigation>
            <OnError/>
              <LogEntry Value="'Konnte Event nicht löschen' + varLastError"
                  Type="error"/>
            </FetchError>

          <!-- Jetzt werden alle lokal geänderten und neu erstellten auf den Server
gepusht -->
          <ELSE Condition="status &lt; 2"/>
            <FetchError BisUpdate="no">
              <SetVar Name="varTitle" BisAttribute="Name"/>
              <SetVar Name="varDescription"
BisAttribute="bisB_Description"/>
              <SetVar Name="varLocation"
BisAttribute="bisB_CalDavLocation"/>
              <IF Condition="STRLEN(varTitle) &gt; 0">
```

```
                <SetVar Name="varTitle" Expression="'SUMMARY:' + varTitle + nl"/>
              <ELSE/>
                <SetVar Name="varTitle" Expression="''"/>
              </IF>
              <IF Condition="STRLEN(varDescription) &gt; 0">
                <SetVar Name="varDescription" Expression="'DESCRIPTION:' + var-
Description + nl"/>
              <ELSE/>
                <SetVar Name="varDescription" Expression="''"/>
              </IF>
              <IF Condition="STRLEN(varLocation) &gt; 0">
                <SetVar Name="varLocation" Expression="'LOCATION:' +
varLocation + nl"/>
              <ELSE/>
                <SetVar Name="varLocation" Expression="''"/>
              </IF>
              <SetVar Name="calendarEvent"/>
              <!-- Organisator -->
              <Navigation>
                VIA bisB_CalDavEventToPerson
              </Navigation>
              <SetVar Name="persCount" BisAttribute="COUNT"/>
              <SetVar Name="varOrganizer" Expression="''"/>
              <IF Condition="persCount &gt; 0">
                <SetVar Name="varOrganizer"
Expression="'ORGANIZER;CN=&quot;' + ATTRIBUTE('Object_Display_Kontext') +
'&quot;:mailto:' +
                  ATTRIBUTE('Comm_Addr_Email') + nl"/>
              </IF>
              <GetVar Name="calendarEvent"/>
              <!-- Bestätigte Teilnehmer -->
              <Navigation>
                VIA bisB_CalDavEventToConfirmedAttendee
              </Navigation>
              <SetVar Name="persCount" BisAttribute="COUNT"/>
              <SetVar Name="varConfAttendees" Expression="''"/>
              <ForEachObject>
                <SetVar Name="varConfAttendees"
Expression="varConfAttendees + 'ATTENDEE;CN=&quot;' +
ATTRIBUTE('Object_Display_Kontext') + '&quot;;PARTSTAT=ACCEPTED:mailto:' + ATTRIB-
UTE('Comm_Addr_Email') + nl"/>
              </ForEachObject>
              <GetVar Name="calendarEvent"/>
              <!-- Unbestätigte Teilnehmer -->
              <Navigation>
                VIA bisB_CalDavEventToAttendee
              </Navigation>
              <SetVar Name="persCount" BisAttribute="COUNT"/>
              <SetVar Name="varAttendees" Expression="''"/>
              <ForEachObject>
                <SetVar Name="varAttendees" Expression="varAttendees +
'ATTENDEE;CN=&quot;' + ATTRIBUTE('Object_Display_Kontext') + '&quot;:mailto:' +
ATTRIBUTE('Comm_Addr_Email') + nl"/>
              </ForEachObject>
```

```
              <GetVar Name="calendarEvent"/>
              <!-- Informationen aus dem bisB_CalDavSyncRemarks, die auch an den
Server gesendet werden müssen -->
              <SetVar Name="syncRemarks" BisAttribute="bisB_CalDavSyncRemarks"/>
              <SetVar Name="syncRemarks" Expression="SUBSTR(syncRemarks,
STRPOS('//////////', syncRemarks, 1) + 10, STRLEN(syncRemarks) -
STRPOS('//////////', syncRemarks, 1) - 9) + nl"/>

              <SetVar Name="varAllDay"
BisAttribute="bisB_CalDavIsAllDayEvent"/>
              <CalDavEncodeDateTime EncodedTime="dtstart" AllDay="varAllDay" Decod-
edTime="ATTRIBUTE('bisT_StartDateTime')"
KeyStr="'DTSTART'"/>
              <CalDavEncodeDateTime EncodedTime="dtend" AllDay="varAllDay" Decoded-
Time="ATTRIBUTE('bisT_EndDateTime')" KeyStr="'DTEND'"/>
              <CalDavEncodeDateTime EncodedTime="dtstamp"
DecodedTime="NOW"/>

              <SetVar Name="iCalString" Expression="
                  'BEGIN:VCALENDAR' + nl +
                  'VERSION:2.0' + nl +
                  'PRODID:-//Byron Informatik AG//BISXDS Scripting' + nl +
                  'BEGIN:VEVENT' + nl +
                  dtstart + nl +
                  dtend + nl +
                  'DTSTAMP:' + dtstamp + nl +
                  varTitle +
                  varDescription +
                  varLocation +
                  varOrganizer +
                  varAttendees +
                  varConfAttendees +
                  syncRemarks +
                  'END:VEVENT' +nl+
                  'END:VCALENDAR'"/>
            <OnError/>
              <LogEntry Value="'Fehler beim Zusammenstellen der Kalendereintrags-
Infos' + varLastError"
                  Type="error"/>
            </FetchError>

            <FetchError>
              <CalDavPushEvent FileName="dbFilename"
                  ICalString="iCalString"
                  EtagVar="updEtag"/>
            <OnError/>
              <LogEntry Value="'Konnte Event nicht auf Server schreiben'
                  + varLastError" Type="error"/>
            </FetchError>

            <!-- aktualisiere die Lookup-Struktur mit dem neuen Etag, sonst wird
der gepushte Event weiter unten noch einmal gepullt -->
            <AddLookup Name="ServerEvents" KeyValue="dbFilename"
StoreValue="updEtag" Mode="replace"/>
```

```
            <FetchError BisUpdate="yes">
              <Navigation>
                MODIFY bisB_CalDavEtag '=updEtag' -- update den geänderten Etag,
der vom Server gemeldet wird
                MODIFY bisB_CalDavStatus 2 -- synchron mit Server
              </Navigation>
            <OnError/>
              <LogEntry Value="'Konnte Event nicht ändern' + varLastError"
                  Type="error"/>
            </FetchError>


          <!-- Alle zum Löschen vormarkierten vom Server löschen -->
          <ELSE Condition="status = 4"/>

            <!-- lokal gelöscht und auf dem Server geändert ist auch ein Synchroni-
sationsproblem -->
            <IF Condition="(etag &lt;&gt; NULL) AND (etag &lt;&gt; dbEtag)">
              <FetchError BisUpdate="yes">
                <Navigation>
                  MODIFY bisB_CalDavStatus 3 -- Synchronisationsproblem
                </Navigation>
              <OnError/>
                <LogEntry Value="'Konnte Event nicht ändern' +
varLastError"
                    Type="error"/>
              </FetchError>
            <ELSE/>
              <FetchError>
                <CalDavDeleteEvent FileName="filename"/>
              <OnError/>
                <LogEntry Value="'Konnte Event nicht vom Server löschen'
                    + varLastError" Type="error"/>
              </FetchError>

              <FetchError BisUpdate="yes">
                <Navigation>
                  DELETE -- kann jetzt auch lokal endgültig gelöscht werden
                </Navigation>
              <OnError/>
                <LogEntry Value="'Konnte Event nicht löschen' +
varLastError"
                    Type="error"/>
              </FetchError>

              <!-- Löschen auf dem Server in der Lookup nachführen -->
              <AddLookup Name="ServerEvents"
                  KeyValue="filename"
                  StoreValue="NULL"
                  Mode="replace"/>
            </IF>
          </IF>
        </ForEachObject>
```

```
        <SetVar Name="changedEventList" Value=""/>
        <FetchError BisUpdate="no">
          <ForEachLookup Name="ServerEvents" KeyVar="filename"
ValueVar="etag">
            <IF Condition="STRLEN(etag) &gt; 0">
              <SetVar Name="UpdateNecessary" Value="FALSE"/>
               <Navigation>
                START VALUE bisB_CalDavFileName = '=filename'
              </Navigation>
              <SetVar Name="count" BisAttribute="COUNT"/>

              <SetVar Name="status"
                  Format="AsInteger"
                  BisAttribute="bisB_CalDavStatus"/>

              <IF Condition="count = 0">
                <AddLookup Name="EventsMissingLocally" KeyValue="filename"
StoreValue="etag"/>
                <SetVar Name="UpdateNecessary" Value="TRUE"/>
              <ELSE Condition="status = 2"/>
                <SetVar Name="dbEtag" BisAttribute="bisB_CalDavEtag"/>

                <!-- Falls Serverversion geändert, sprich Etag anders -->
                <IF Condition="dbEtag &lt;&gt; etag">
                  <SetVar Name="UpdateNecessary" Value="TRUE"/>
                </IF>
              </IF>
            </IF>

            <!-- Für Multiget: erstelle Liste aller Events, die in einem Zug vom
Server geholt werden sollen -->
            <IF Condition="UpdateNecessary = TRUE">
              <SetVar Name="changedEventList"
                  Expression="changedEventList + filename + #9"/>
            </IF>
          </ForEachLookup>
        <OnError/>
          <LogEntry Value="'Problem beim Auflisten der Server-Events' + varLastEr-
ror"
              Type="error"/>
        </FetchError>

        <FetchError BisUpdate="yes">
          <ForEachLookup Name="EventsMissingLocally" KeyVar="filename"
ValueVar="etag">
            <Navigation>
              CREATE 1 bisB_CalDavEvent ()
              MODIFY bisB_CalDavFilename DATAOF filename
              MODIFY bisB_CalDavEtag DATAOF etag
              MODIFY bisB_CalDavStatus 3 -- temporär bis alle Details vom
                                         -- Server geholt wurden
            </Navigation>
          </ForEachLookup>
        <OnError/>
```

```
          <LogEntry Value="'Konnte Event nicht erzeugen' + varLastError"
              Type="error"/>
        </FetchError>

        <FetchError>
          <CalDavMultiget FileNames="changedEventList" FileNameVar="filename" Ex-
pandRecurringFrom="NOW" ExpandRecurringUntil="NOW + 365">
            <FetchError BisUpdate="yes">
              <GetLookup Name="ServerEvents" KeyValue="filename"
ToVar="etag"/>
              <Navigation>
                START VALUE bisB_CalDavFileName = '=filename'
                MODIFY bisB_CalDavStatus 3 -- Synchronisationsproblem
                MODIFY bisB_CalDavEtag DATAOF etag
                MODIFY bisB_CalDavSyncRemarks "//////////"
              </Navigation>
            </FetchError>

            <SetVar Name="currentlyReading" Value="unknown"/>
            <FetchError BisUpdate="yes">
              <CalDavFetchEventDetails RecNameVar="recName"
                RecValueVar="recValue">
                <SetVar Name="calendarEvent"/>

                <IF Condition="recName = 'BEGIN'">
                  <SetVar Name="currentlyReading" Expression="recValue"/>
                <ELSE Condition="recName = 'END'"/>
                  <IF Condition="currentlyReading = 'VALARM'">
                    <SetVar Name="currentlyReading" Value="VEVENT"/>
                  <ELSE/>
                    <SetVar Name="currentlyReading" Value="unknown"/>
                  </IF>
                </IF>

                <IF Condition="currentlyReading = 'VEVENT'">
                  <IF Condition="recName = 'SUMMARY'>
                    <Navigation>
                      MODIFY Name DATAOF recValue
                    </Navigation>
                  <ELSE Condition="recName = 'DESCRIPTION'"/>
                    <Navigation>
                      MODIFY bisB_Description DATAOF recValue
                    </Navigation>
                  <ELSE Condition="recName = 'LOCATION'"/>
                    <Navigation>
                      MODIFY bisB_CalDavLocation DATAOF recValue
                    </Navigation>
                  <ELSE Condition="recName = 'DTSTART'"/>
                    <CalDavDecodeDateTime DecodedTime="dtstart"
                        EncodedTime="recValue" AllDay="allDay"/>
                    <Navigation>
                      MODIFY bisT_StartDateTime DATAOF dtstart
                      MODIFY bisB_CalDavIsAllDayEvent '=allDay'
                    </Navigation>
```

```
                  <ELSE Condition="recName = 'DTEND'"/>
                    <CalDavDecodeDateTime DecodedTime="dtend"
                        EncodedTime="recValue"/>
                    <Navigation>
                      MODIFY bisT_EndDateTime DATAOF dtend
                    </Navigation>
                  <ELSE Condition="recName = 'ORGANIZER'"/>
                    <SetVar Name="mailAddr" Expression="SUBSTR(recValue,
STRPOS('mailto:', recValue, 1) + 7, STRLEN(recValue) - STRPOS('mailto:', recValue,
1) - 6)"/>
                    <Navigation>
                      START VALUE Comm_Addr_EMail = DATAOF mailAddr
                    </Navigation>
                    <SetVar Name="person"/>
                    <SetVar Name="count" BisAttribute="COUNT"/>
                    <GetVar Name="calendarEvent"/>
                    <IF Condition="count = 0">
                      <Navigation>
                        MODIFY bisB_CalDavSyncRemarks "='Unbekannter Organisator '
+ mailAddr + nl +
ATTRIBUTE('bisB_CalDavSyncRemarks') + nl + recName + ';' + recValue"
                      </Navigation>
                    <ELSE/>
                      <Navigation>
                        BIND bisB_CalDavEventToPerson (RECALL person)
REBIND
                      </Navigation>
                    </IF>
                  <ELSE Condition="recName = 'ATTENDEE'"/>
                    <SetVar Name="mailAddr" Expression="SUBSTR(recValue,
STRPOS('mailto:', recValue, 1) + 7, STRLEN(recValue) - STRPOS('mailto:', recValue,
1) - 6)"/>
                    <Navigation>
                      START VALUE Comm_Addr_EMail = DATAOF mailAddr
                    </Navigation>
                    <SetVar Name="person"/>
                    <SetVar Name="count" BisAttribute="COUNT"/>
                    <GetVar Name="calendarEvent"/>
                    <IF Condition="count = 0">
                      <Navigation>
                        MODIFY bisB_CalDavSyncRemarks "='Unbekannter Teilnehmer ' +
mailAddr + nl + ATTRIBUTE('bisB_CalDavSyncRemarks') + nl + recName + ';' +
recValue"
                      </Navigation>
                    <ELSE/>
                      <IF Condition="STRPOS('PARTSTAT=ACCEPTED', recValue, 0) &gt;
0">
                        <Navigation>
                          BIND bisB_CalDavEventToConfirmedAttendee (RECALL person)
                        </Navigation>
                      <ELSE/>
                        <Navigation>
                          BIND bisB_CalDavEventToAttendee (RECALL person)
                        </Navigation>
```

```
                    </IF>
                  </IF>
                <ELSE Condition="(recName &lt;&gt; 'BEGIN') AND (recName &lt;&gt;
'END') AND (recName &lt;&gt; 'UID') AND (recName &lt;&gt; 'DTSTAMP')"/>
                  <SetVar Name="remarks" Expression="'Bisher unimplementierte
Eigenschaft ' + recName + '=' + recValue + nl + ATTRIB-
UTE('bisB_CalDavSyncRemarks')"/>
                  <Navigation>
                    MODIFY bisB_CalDavSyncRemarks '=remarks'
                  </Navigation>
                </IF>
              <ELSE Condition="currentlyReading = 'VALARM'"/>
                <!--Navigation>
                  [SAVEAS event VIA Object_To_Children CLASS bisB_CalDavReminder
ELSIF=0 CREATE 1 bisB_CalDavReminder (RECALL event)]
                  RECALL event
                </Navigation-->
                <IF Condition="recName = 'TRIGGER'">
                  <!-- Siehe http://tools.ietf.org/html/draft-daboo-valarm-
extensions-04#page-13 -->
                  <IF Condition="recValue &lt;&gt; 'VALUE=DATE-
TIME:19760401T005545Z'">
                    <Navigation>
                      MODIFY bisB_CalDavSyncRemarks "='Erinnerung ' + recValue +
nl + ATTRIBUTE('bisB_CalDavSyncRemarks') + nl + 'BEGIN:VALARM' + nl + recName + ';'
+ recValue + nl + 'END:VALARM'"
                    </Navigation>
                  </IF>
                <ELSE Condition="(recName &lt;&gt; 'BEGIN') AND (recName &lt;&gt;
'END') AND (recName &lt;&gt; 'UID') AND (recName &lt;&gt; 'DTSTAMP')"/>
                  <SetVar Name="remarks" Expression="'Bisher unimplementierte
Valarm-Eigenschaft ' + recName + '=' + recValue + nl +
ATTRIBUTE('bisB_CalDavSyncRemarks')"/>
                  <Navigation>
                    MODIFY bisB_CalDavSyncRemarks '=remarks'
                  </Navigation>
                </IF>
              </IF>

            </CalDavFetchEventDetails>
            <Navigation>
              MODIFY bisB_CalDavStatus 2 -- synchron mit Server
            </Navigation>
          <OnError/>
            <LogEntry Value="'Fehler beim Schreiben der Events in die Datenbank'
+
                varLastError" Type="error"/>
          </FetchError>

        </CalDavMultiget>
      <OnError/>
        <LogEntry Value="'Fehler beim Lesen von Kalenderdaten vom Server (Multi-
get)' +
            varLastError" Type="error"/>
```

```
        </FetchError>

      </CalDavConnection>
    <OnError/>
      <LogEntry Value="'Unbekannter Fehler' + varLastError" Type="error"/>
    </FetchError>
  </Script>
</Defs>
```