LibreOffice

/libreoffice.org

ask.libreoffice.org

LibreOffice
Dokumentationsteam

XML-Formulare

Erstellen Sie XML-Formulardokumente

Version: 6.0

Weitere Informationen finden Sie unter https://de.libreoffice.org

Copyright

Autoren

Robert Großkopf

Mitwirkende

Klaus-Jürgen Weghorn

Rückmeldung (Feedback)

Kommentare oder Vorschläge zu diesem Dokument können Sie in deutscher Sprache an die Adresse *discuss@de.libreoffice.org* senden.

Vorsicht 	Alles, was an eine Mailingliste geschickt wird, inklusive der E-Mail-Adresse und anderer persönlicher Daten, die die E-Mail enthält, wird öffentlich archiviert und kann nicht gelöscht werden. Also, schreiben Sie mit Bedacht!

Datum der Veröffentlichung und Softwareversion

Veröffentlicht am 10.4.2018. Basierend auf der LibreOffice Version 6.0.

Inhalt

Einleitung

Mit XML-Formulardokumenten werden Formulare erstellt, die XML-Dateien erzeugen.Diese werden in ihrer entsprechenden Struktur mit Daten versehen, die z.B. lokal gespeichert werden. Außerdem kann so ein Formular dazu genutzt werden, Daten in einer XML-Datensammlung zu speichern und über einen Browser in einer sinnvollen Übersicht auszugeben.

Für das Ausführen eines mit LibreOffice erstellten XML-Formulars ist eine Installation von LibreOffice erforderlich. Mit anderen Programmen als den LibreOffice verwandten Office-Programmen ist die Nutzung von XML-Formulardokumenten nicht möglich.

In dieser Dokumentation geht es darum, zuerst einmal ein einfaches Formular erstellen zu können. Anschließend wird gezeigt, wie die Inhalte eines solchen Formulars weiter genutzt werden können und zu größeren Datensammlungen, auch über das Internet mit einem Server, zusammengefasst werden können.

Im Gegensatz zu einfachen Formularen kann die Eingabe der Daten mit XML-Formularen für jedes Feld sehr genau eingegrenzt werden, ohne dass dafür, wie z.B. in Base, der Einsatz von Makros erforderlich wäre. Auch Berechnungen während der Eingabe sind problemlos möglich. Dies wird anschließend anhand eines komplexeren Formulars erklärt, das möglichst viele der zur Verfügung stehenden Funktionen der XML-Formulare nutzt.

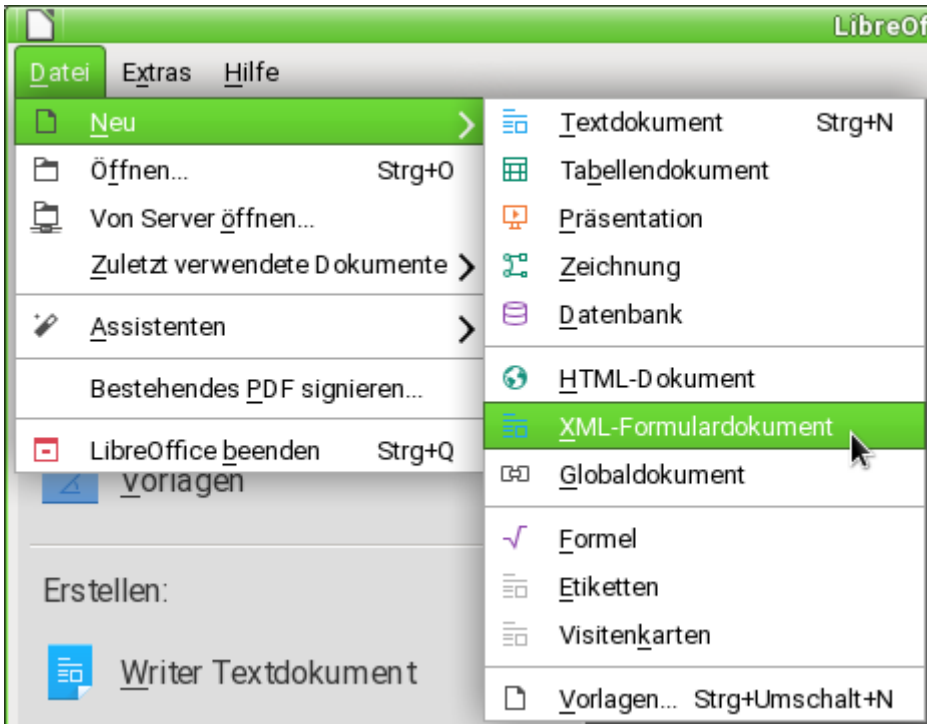
Tipp	Um XML-Formulardokumente erstellen und verwalten zu können – und damit Grundlagenwissen dieser Dokumentation – sind das Verständnis und die sichere Handhabung von: • Erstellen und Bedienen von Formularen in LibreOffice • Beherrschen von Formularsteuerelementen in LibreOffice • Erstellen und Bedienen von Makros in LibreOffice. Sollten Sie hier noch nicht so firm sein, finden Sie weitere Hinweise in den Erste-Schritte-Handbüchern sowie im Base- und im Macro-Handbuch: *https://de.libreoffice.org/get-help/documentation/*

Hinweis	XML-Dateien, also auch die Daten, die über XML-Formulare verarbeitet werden, sind nicht weiter gesichert. Die Dateien sind im Klartext lesbar, können im Browser übersichtlich dargestellt werden und enthalten standardmäßig keinen weiteren Schutz. Bei Formularen, die den Inhalt über das Netz verschicken, muss der Schutz entsprechend in die Auswertung eingebaut werden. Werden Daten über Email weitergeleitet, so sollten die Mails möglichst verschlüsselt werden.

Hinweis	XML-Dateien, die das Formular erzeugt, haben den folgenden Aufbau:[1]
	`<?xml version="1.0" encoding="UTF-8"?>`
	`<instanceData>`
	`  <Vorname>Lieschen</Vorname>`
	`  <Nachname>Müller</Nachname>`
	`</instanceData>`
	Einem einführenden Tag, der die Datei als XML-Datei kennzeichnet und zusätzlich noch den verwendeten Zeichensatz enthält, folgt ein Element, das die gesamten Daten umfasst. In der Regel gibt es zu jedem Element einen Starttag und einen Endtag, wobei der Endtag an dem / zu erkennen ist. Text, der zwischen diesen Tags steht, enthält die Daten, hier also den Vornamen «Lieschen» und den Nachnamen «Müller».

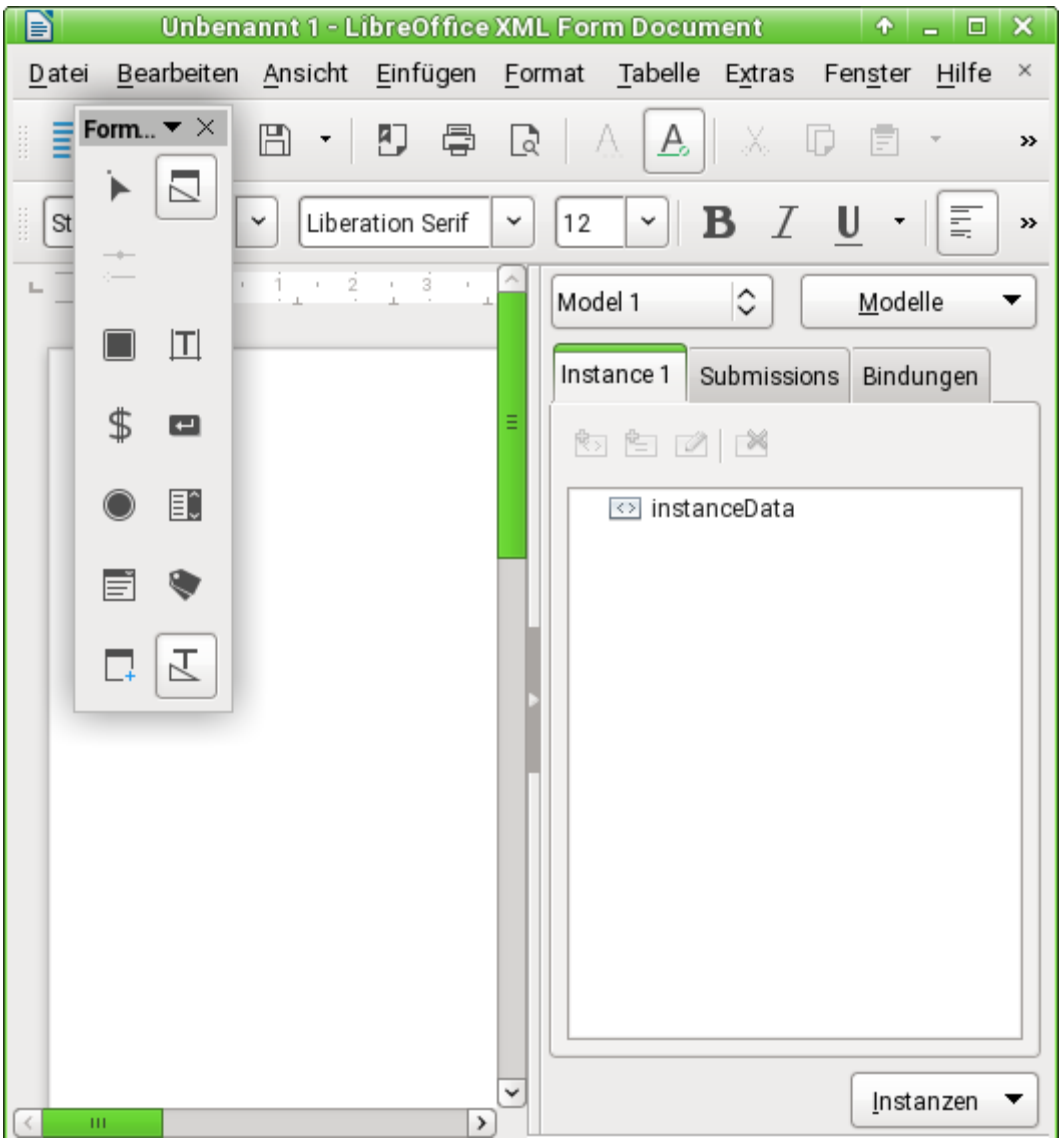
Ein XML-Formular Schritt für Schritt

LibreOffice wird gestartet.

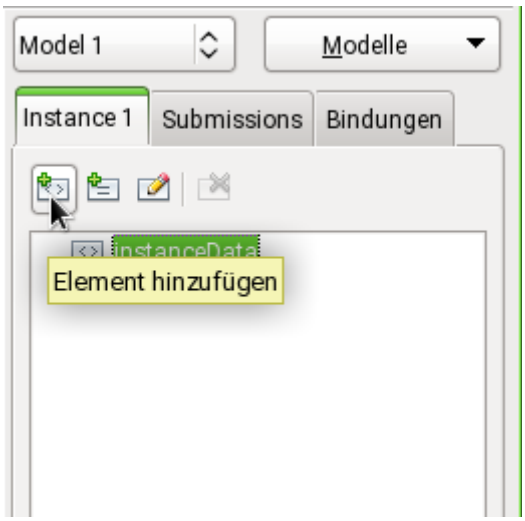


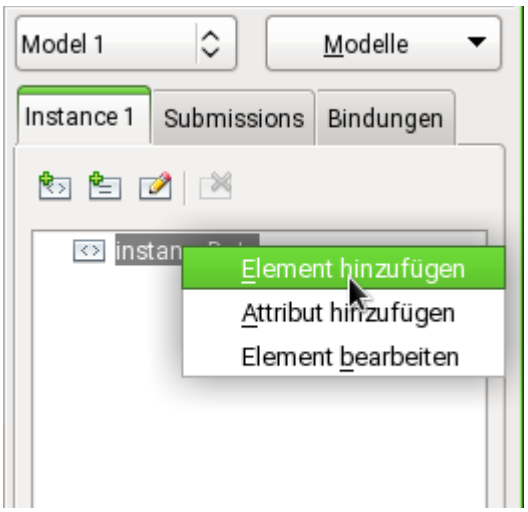
Über **Datei → Neu → XML-Formulardokument** wird eine Writeransicht zusammen mit der für XML-Formulare erforderlichen Seitenleiste erstellt.

1 Siehe: *https://de.wikipedia.org/wiki/Extensible_Markup_Language*

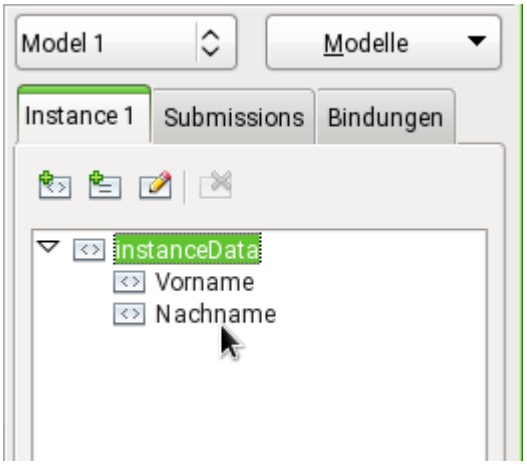


Wird die Symbolleiste für die Formular-Steuerelemente nicht angezeigt, so kann diese über **Ansicht → Symbolleisten → Formular-Steuerelemente** eingeblendet werden.

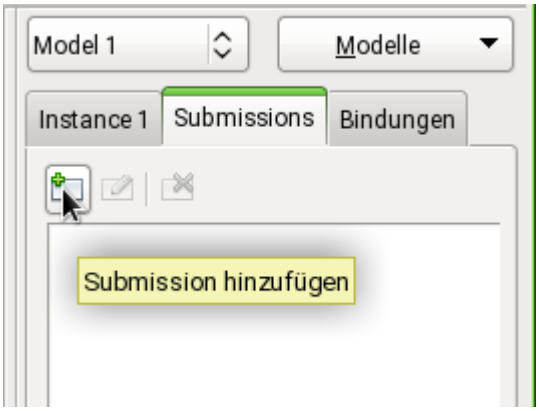




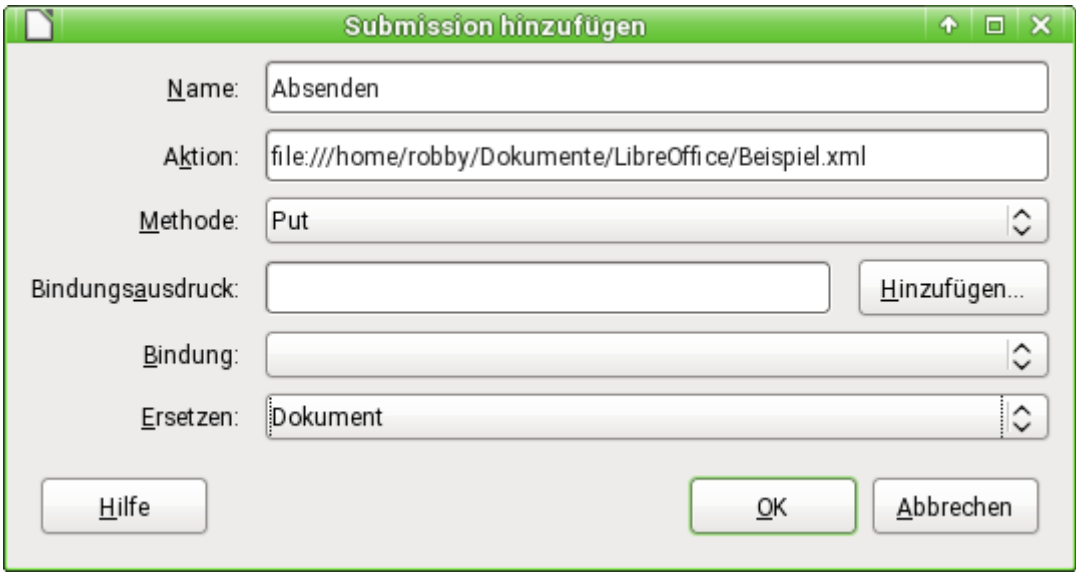
In dem Reiter **Instance 1** liegt bereits das umfassende Element **instanceData**. Diesem Element werden im folgenden zwei gleichberechtigte Elemente hinzugefügt.



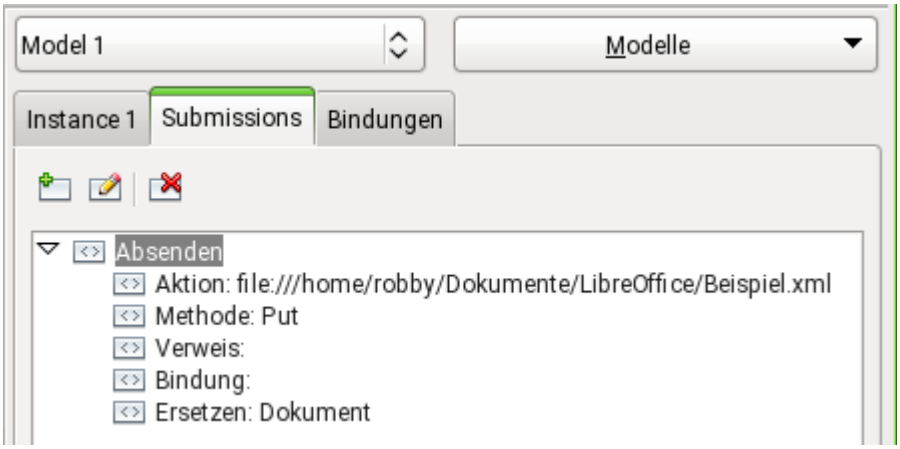
Der dabei erscheinende Dialog wird jeweils nur in der Eigenschaft **Name** bearbeitet. Hier sind die Untergliederungen «Vorname» und «Nachname» hinzugefügt worden. Dadurch ist die komplette durch dieses Formular zu erstellende XML-Datei definiert.



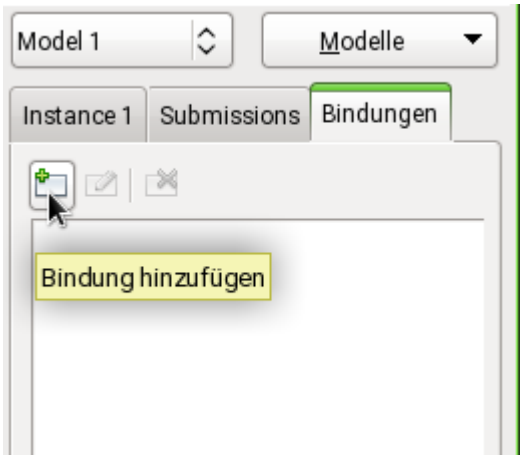
Im Reiter **Submissions** wird eine Submission hinzugefügt.



Die Submission wird mit dem **Namen** «Absenden», dem Pfad für die **Aktion** auf dem lokalen Rechner und der **Methode** «Put» (siehe: PUT-Methode) versehen. Bei «Absenden» wird das vorherige Dokument «Beispiel.xml» mit dem neuen Inhalt überschrieben: **Ersetzen → Dokument**.



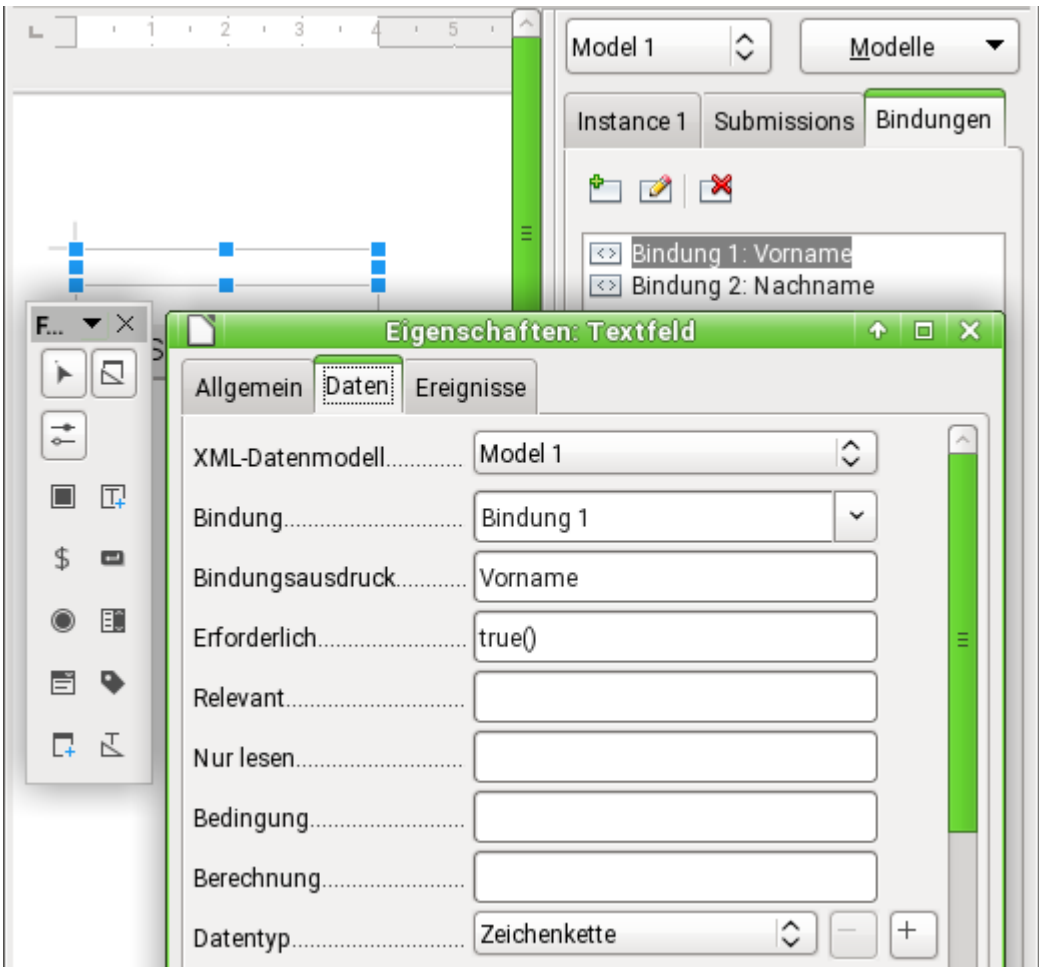
Die entsprechenden Einträge für die Submission lassen sich unter dem Reiter **Submissions** anzeigen.



Jetzt wird eine Bindung hinzugefügt, die den Angelpunkt zu den geplanten Formularfeldern darstellt. Nur so ist es möglich, die Formularfelder mit den entsprechenden Instanzen zu koppeln.

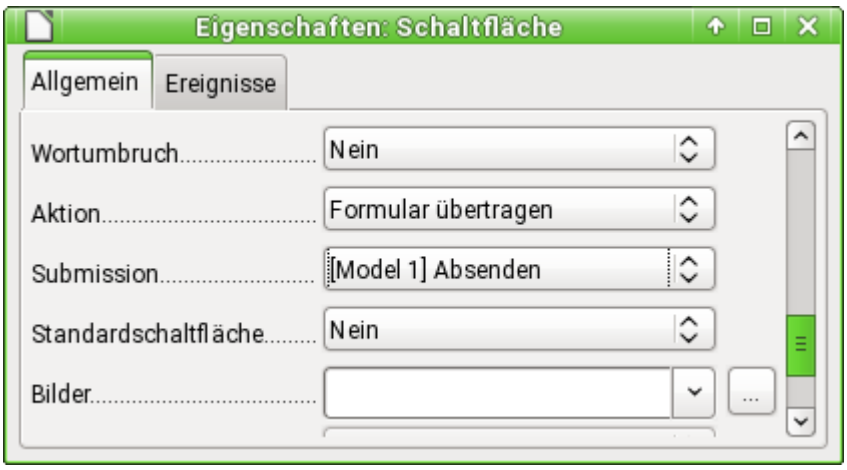
Die Bindungen werden durchnummeriert, können natürlich auch mit separaten Namen versehen werden. Wird hier der gleiche Ausdruck wie bei den Instanzen als Bindungsausdruck eingegeben, so erfolgt die entsprechende Koppelung zu den Instanzen.

Jede Bindung wird mit entsprechenden Eigenschaften versehen. Diese können aber auch bereits bei der Festlegung der Instanzen mit dem gleichen Dialog erstellt werden. Gegebenenfalls können sie auch noch in den Formularfeldern nachgetragen werden.



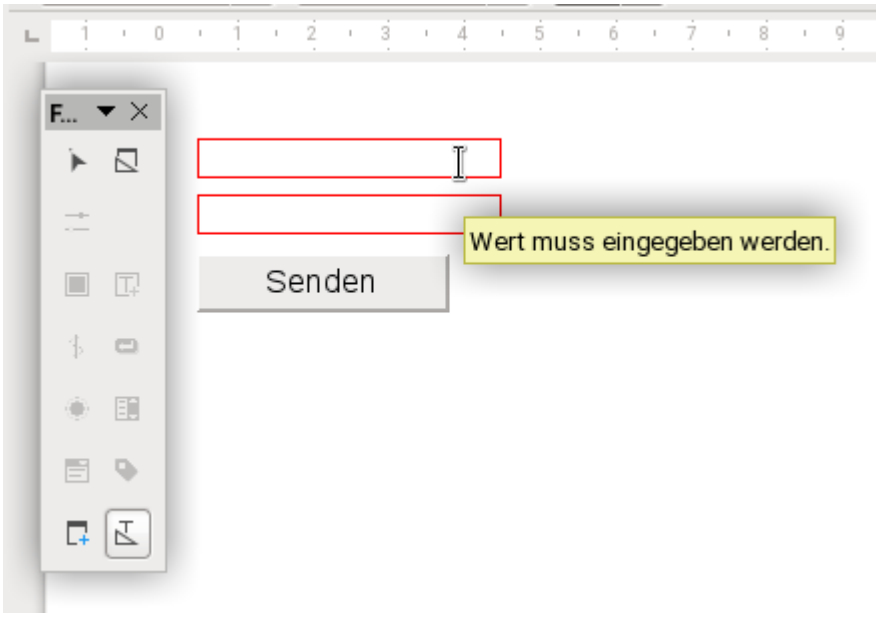
Aus den Formular-Steuerelementen[2] wird das Textfeld ausgesucht. Im Gegensatz zu Datenbanken erfolgt jetzt im Reiter **Daten** eine Verknüpfung zu dem Datenmodell.

In diesem Beispiel steht nur das **Model 1** zur Verfügung. **Bindung** lässt mit einem Auswahlfeld die Auswahl der beiden definierten Bindungen zu, hier also von **Bindung 1**. Die restlichen Einträge spiegeln die Definition der Bindung wieder.

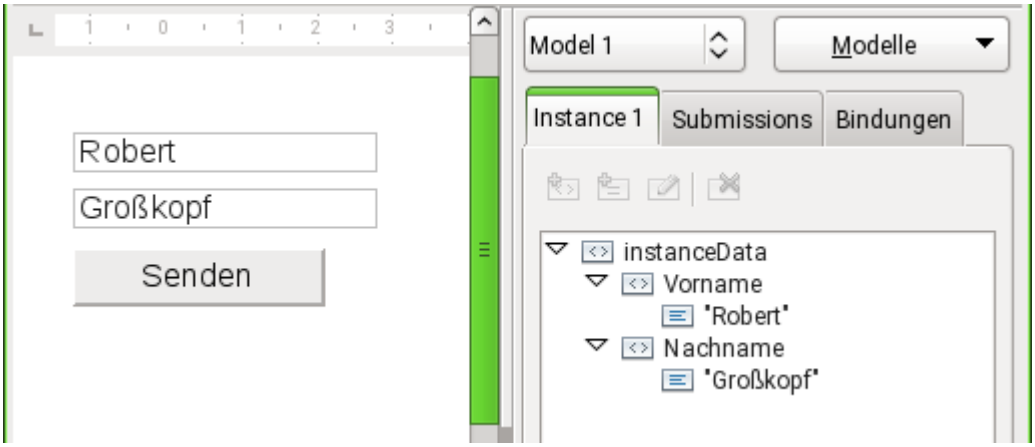


Dem Formular wird ein Button hinzugefügt. In den allgemeinen Eigenschaften wird hier die **Aktion → Formular übertragen** ausgewählt. In dem Feld **Submission** ist jetzt **[Model 1] Absenden** verfügbar.

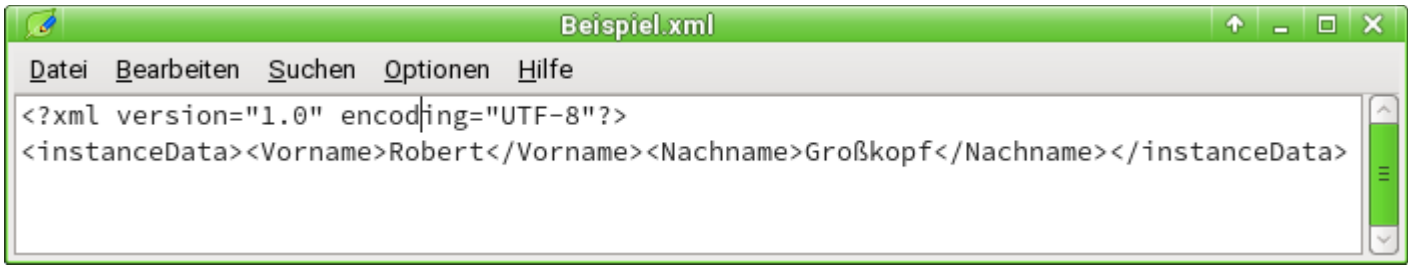
2 Zu Formularsteuerelementen siehe auch das Erste-Schritte-Handbuch sowie das Base-Handbuch: *https://de.libreoffice.org/get-help/documentation/*



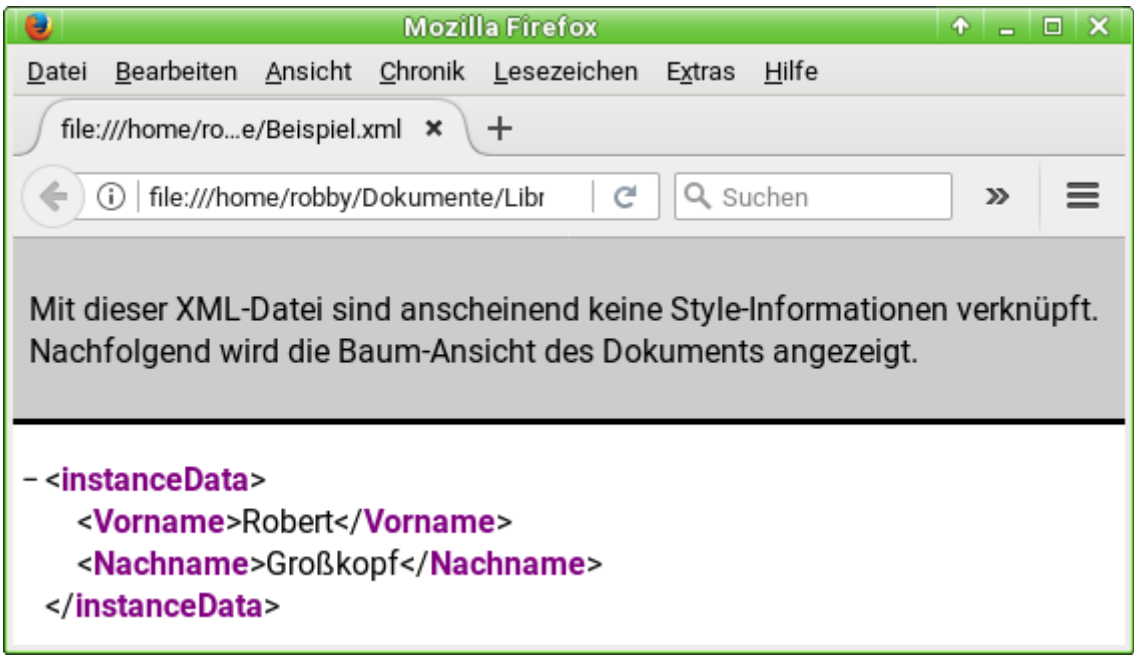
Wird jetzt über die Symbolleiste der Entwurfsmodus des Formulars ausgeschaltet, so ist die Eingabe von Daten in das Formular möglich. Durch die rote Umrandung und die Zusatzinformation weist das Formular darauf hin, dass in den Feldern auf jeden Fall eine Eingabe erforderlich ist.



Nach erfolgter Eingabe und dem Absenden bleiben die Daten in dem Formular stehen. Unter dem Reiter **Instance 1** sind jetzt die Inhalte der Untergliederungen **Vorname** und **Nachname** zu erkennen. Die Daten wurden in die definierte XML-Datei geschrieben und über das Formular wieder ausgelesen. Sie bleiben auch in dem Formulardokument weiter erhalten. Das Formular zurücksetzen kann hier nur ein Makro.



Die XML-Datei «Beispiel.xml» weist zwei Zeilen auf. Sie startet mit der Definitionszeile und schreibt dann alle folgenden Einträge des Elements «instanceData» in eine Zeile.



Wird die Datei mit Firefox geöffnet, so erkennt Firefox den Aufbau der XML-Datei und stellt sie entsprechend gegliedert dar. Firefox fehlen hier lediglich Style-Informationen, die statt der Baum-Ansicht eine nutzerfreundlichere Ansicht bieten.

Datensammlung formatiert

Schöner wäre es natürlich, wenn Firefox Style-Informationen erhalten würde und nicht nur die aktuellen Daten gesammelt würden. Das sähe dann beispielsweise so aus:

Um dies zu bewerkstelligen muss an unterschiedlichen Stellen nachgearbeitet werden.

Erstellen einer XSL-Datei zur Formatierung

Die XML-Datei bezieht seine Formatierungen aus einer Datei mit separaten Formatanweisungen. Die Datei «Daten_Start_formatiert.xsl» hat die folgende Struktur:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <HTML>
         <Head>
            <Title>XML-Formular</Title>
         </Head>
         <Body>
```

```
                <H2>Daten aus dem XML-Formular</H2>
                <table>
                    <tr>
                        <th>Nachname</th>
                        <th>Vorname</th>
                    </tr>
                    <xsl:for-each select="Data/instanceData">
                    <xsl:sort select="Nachname" order="ascending" data-type="text" />
                    <tr>
                        <td><xsl:value-of select="Nachname"/></td>
                        <td><xsl:value-of select="Vorname"/></td>
                    </tr>
                    </xsl:for-each>
                </table>
            </Body>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```

Die Datei enthält innerhalb der Tags, die für die XSL-Datei notwendig sind, einen Aufbau wie eine einfache HTML-Datei. Diese HTML-Datei besteht aus einer Überschrift und einer Tabelle mit den Spaltentiteln «Nachname» und «Vorname».

In einer **`xsl:for-each`**-Schleife wird aus der XML-Datei alles ausgelesen, was sich innerhalb des Bereiches **`instanceData`** befindet. Da instanceData mehrmals vorkommt, muss hier die Schleife erfolgen.

Mit der **`xsl:sort`**-Anweisung werden die Daten, die sich in den instanceData-Bereichen befinden, nach dem **`Nachname`** aufsteigend sortiert.

Aus den Feldern **`Nachname`** und **`Vorname`** innerhalb von **`instanceData`** werden die entsprechenden Werte ausgelesen und in der Tabellenzeile dargestellt.

Die XML-Datei «Daten_Start_formatiert.xml» weist den folgenden Inhalt auf:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="Daten_Start_formatiert.xsl" ?>
<Data>
<instanceData><Vorname>Robert</Vorname><Nachname>Großkopf</Nachname></instanceData>
<instanceData><Vorname>Jutta</Vorname><Nachname>Schön</Nachname></instanceData>
<instanceData><Vorname>Ammon</Vorname><Nachname>Düül</Nachname></instanceData>
<instanceData><Vorname>Nick</Vorname><Nachname>Knatterton</Nachname></instanceData>
<instanceData><Vorname>Silvia</Vorname><Nachname>Bach</Nachname></instanceData>
<instanceData><Vorname>Rob</Vorname><Nachname>van Buur</Nachname></instanceData>
<instanceData><Vorname>Bob</Vorname><Nachname>Baumeister</Nachname></instanceData>
</Data>
```

In der zweiten Zeile dieser Datei wird darauf hingewiesen, dass die Darstellung in einer separaten Datei «Daten_Start_formatiert.xsl» definiert ist. Die Hauptinstanz **`<Data>`** darf hier nur einmal vorkommen, wie auch **`<instanceData>`** nur einmal in «Daten.xml» vorkommt.

Diese XML-Datei wird mit Hilfe eines Makros mit neuen Datensätzen aus der Datei «Daten_Start.xml» versorgt, die jeweils als neuer Tag <**`instanceData`**> als letzter Datensatz angehängt werden. Die Datei «Daten_Start.xml» ist dabei die, die das Formular standardmäßig über die Einträge von **Submissions** erstellt.

Kopieren der neuen Daten in eine dauerhafte Datendatei

Eine Kopie der Daten kann entweder händisch oder über ein Makro erfolgen. Das folgenden Makro ist an den Button des Formulars mit dem **Ereignis → Maustaste losgelassen** verknüpft.[3]

```
SUB DatenNachXmlDatei(oEvent AS OBJECT)
    DIM a()
    DIM b()
```

3 Siehe Base-Handbuch, Kapitel «Makros» (*https://de.libreoffice.org/get-help/documentation/*)

```
    stInstance = oEvent.Source.Model.Tag
```

Zuerst werden die Variablen definiert. Hier sind, im Gegensatz zum Quelltext, nur die unbedingt notwendigen Definitionen der Arrays aufgeführt, die zum Auftrennen von Texten benötigt werden.

Anschließend wird aus den Zusatzinformationen (**Tag**) des Buttons der Name der Instanz ausgelesen. In dem Beispiel steht dann hier «Data».

```
    oDoc = ThisComponent
    stDir = Left(oDoc.Location,Len(oDoc.Location)-Len(oDoc.Title))
    stFileTmp = stDir & "Daten_Start.xml"
    stFileData = stDir & "Daten_Start_formatiert.xml"
```

Aus dem aktuellen Formular wird der Pfad ausgelesen, indem einfach der Dateiname abgetrennt wird. Dann werden die Datenquelle «Daten_Start.xml» und das Ziel für die Daten «Daten_Start_formatiert.xml» mit dem Pfad verbunden. Beide Dateien befinden sich also im gleichen Verzeichnis wie die Formulardatei. «Daten_Start.xml» wird allerdings bei jeder Neueingabe vom Formular aus komplett überschrieben. Hier können also nicht Daten hinzugefügt werden.

```
    oFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
    IF oFileAccess.exists(stFileTmp) THEN
        oInputStream = createUnoService("com.sun.star.io.TextInputStream")
        oFile = oFileAccess.OpenFileReadWrite(stFileTmp)
        oInputStream.SetInputStream(oFile.getInputStream)
        DO WHILE NOT oInputStream.isEOF
            stTmp = stTmp & oInputStream.ReadLine()
        LOOP
        oInputStream.closeInput
```

Mit einem Uno-Service wird auf den **SimpleFileAccess** zugegriffen. Mit diesem wird die Datei «Daten_Start.xml» zeilenweise ausgelesen und in der Variablen **stTmp** zusammengefasst. Anschließend wird der Datenfluss beendet.

Die Variable **stTmp** wird aufgesplittet, so dass schließlich nur noch der Inhalt in **stTmpData** übrig bleibt, der in der Datei «Daten_Start.xml» nach dem mit «?>» endenden Tag steht.

```
        a = Split(stTmp,"?>")
        stTmpData = a(1)
```

Wie bei der ersten Datei wird jetzt auch die Datendatei «Beispiel_formatiert.xml» ausgelesen. Allerdings wird der beim Auslesen abgeschnittene Zeilenumbruch jedes Mal wieder mit **CHR(13)&CHR(10)** hinzugefügt.

```
        IF oFileAccess.exists(stFileData) THEN
            oInputStream = createUnoService("com.sun.star.io.TextInputStream")
            oFileData = oFileAccess.OpenFileReadWrite(stFileData)
            oInputStream.SetInputStream(oFileData.getInputStream)
            DO WHILE NOT oInputStream.isEOF
                stDataOld = stDataOld & oInputStream.ReadLine() & CHR(13) & CHR(10)
            LOOP
            oInputStream.closeInput
```

Die in die Datei zu schreibenden Daten werden aus den bestehenden Daten und der neuen Zeile neu zusammengestellt. Die Datei wird anschließend neu geschrieben. Zum Schluss wird die Ausgabe in die Datei wieder geschlossen.

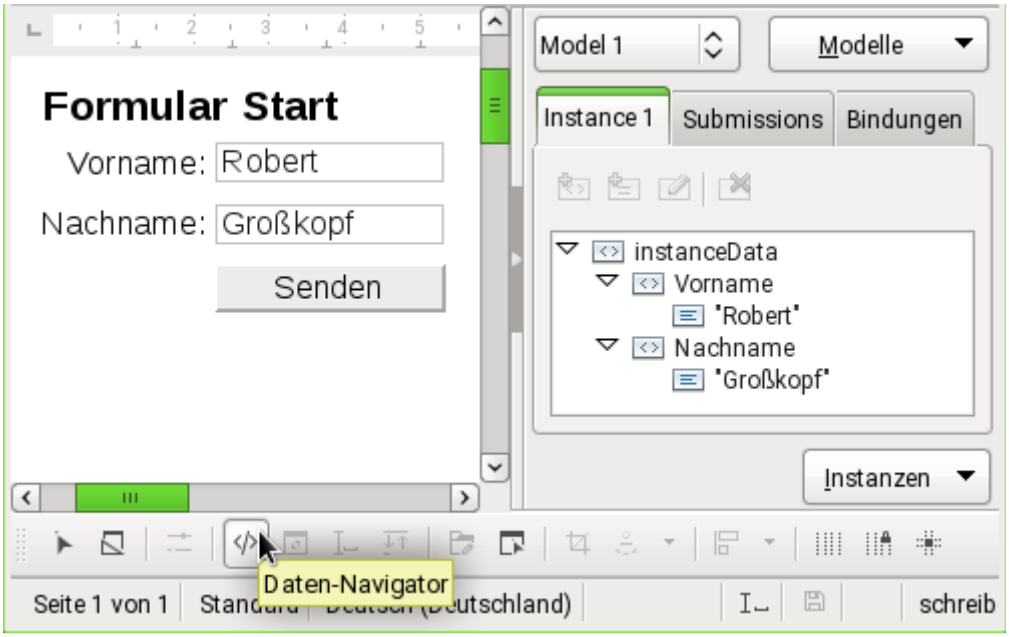
```
            oOutputStream = createUnoService("com.sun.star.io.TextOutputStream")
            b = Split(stDataOld,"</" & stInstance & ">")
            stDataNew = b(0) & stTmpData & CHR(13) & CHR(10) & "</" & stInstance & ">"
            oFileData = oFileAccess.OpenFileReadWrite(stFileData)
            oOutputStream.SetOutputStream(oFileData.getOutputStream)
            oOutputStream.writeString(stDataNew)
            oOutputStream.closeOutput()
        END IF
    END IF
END SUB
```

Wie hier nur am Beispiel des Umlesens der Daten gezeigt wurde kann so etwas natürlich auch zum Einlesen der Daten in eine Datenbank genutzt werden. Die Befüllung der Datenbank würde

dann über ein Formular erfolgen, von dem aus die Person, die die Daten schreibt, nicht zwingend Zugriff auf die Daten hat, die bereits geschrieben wurden.

Formulareingabe optimieren

In der Standardeinstellung wird bei XML-Formulardokumenten immer die Seitenleiste mit den Datenstrukturen des XML-Formulars (**Daten-Navigator**) und die Formular-Steuerelemente angezeigt. Die Symbolleiste für den Formular-Entwurf zeigt gegenüber Base-Formularen ein zusätzliches Symbol, mit der der **Daten-Navigator** ausgeschaltet werden kann:



Der Daten-Navigator ist dann nicht mehr über die Funktion zum Einblenden der Seitenleiste anzeigbar. Wird dann zusätzlich noch die Symbolleiste für den Formularentwurf entfernt, so ist das Formular erst einmal gegen unbedachte Fehlnutzungen geschützt. Änderungen an diesen Stellen bewirken keine Änderungen an dem Dokument und können daher nicht mit dem Formular abgespeichert werden. Sie werden nur in den persönlichen Nutzereinstellungen gespeichert. Wird allerdings die Datei über **Datei → Eigenschaften → Sicherheit → Optionen für Dateifreigabe → Schreibgeschützt öffnen** beim Öffnen mit einem Schreibschutz versehen, so wird das Formular nur zur Eingabe von Daten geöffnet.

Versandmethoden für den Formularinhalt[4]

In dem Reiter **Submissions** sind die Versandmöglichkeiten für den Formularinhalt unter gebracht. Es stehen die für Internetformulare möglichen POST- und GET-Methoden sowie die nur lokal gebräuchliche Methode PUT zur Verfügung. Die POST-Methode wird in dem Editor allgemein als «Senden» bezeichnet.

POST-Methode

Hier wird der gesamte Inhalt als ein Element übertragen. Diese Methode findet Anwendung für größere Formularinhalte oder auch für den Versand per Mail. Unter **Aktion** ist dann einzutragen:

http://www.example.com/xmldata.php

4 Siehe hierzu das Beispiel «Formular_einfach.odt»

mailto:lieschen.mueller@example.org

Der Mailversand läuft leider nicht ordnungsgemäß mit diesem Kommando ab, so dass er nur mit Hilfe eines Makros und der PUT-Methode möglich ist.

Die Methode ruft mit dem Eintrag **Submissions → Ersetzen → Dokument** die ausführende Datei mehrmals auf, dabei ab dem 2. Mal ohne Inhalt. Außerdem öffnet Writer z.B. die Datei «xmldata.php». Wird stattdessen der Eintrag **Submissions → Ersetzen → Kein** gewählt, so kann der Inhalt anschließend durch das Script am Server einwandfrei verarbeitet werden.

Der Empfang der gesandten Daten gestaltet sich etwas problematisch, da der Inhalt der xml-Datei verschickt wird. Die o.g. «xmldata.php» müsste dann mit dem folgenden Befehl auf die Struktur der Datei zugreifen:

```php
<?php
//Datei xmldata.php
$datafile = file_get_contents("php://input");
?>
```

Der Inhalt sieht dann so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<instanceData><Vorname>Lieschen</Vorname><Nachname>Müller</Nachname></instanceData>
```

Sollen die Daten gleich in eine formatierte XML-Datensammlung eingebunden werden, so erfolgt direkt in der PHP-Datei die Weiterverarbeitung:

```php
<?php
//Datei xmldata.php
$datafile = file("php://input");
IF (count($datafile) > 1) {
    $data = $datafile[1];
    IF (file_exists("data/Daten_einfach_formatiert.xml")) {
        $datei = file("data/Daten_einfach_formatiert.xml") ;
        $maxNode = $datei[count($datei)-1];
        array_pop($datei); // letztes Element löschen
        $datei[] = $data."\r\n</".$maxNode.">";
        $datei_alt = fopen("data/Daten_einfach_formatiert.xml", "w+") ;
        foreach($datei as $zeile){
            fwrite($datei_alt, $zeile);
        }
        fclose($datei_alt);
    }
}
?>
```

Zuerst wird der Inhalt über **`file`** in ein Array gelesen. Enthält das Array neben der ersten Zeile Daten, so erfolgt eine Weiterverarbeitung. Der Dateninhalt des Arrays befindet sich in dem 2. Arrayelement, also **`$datafile[1]`**, da die Zählung mit «0» beginnt. Die zu beschreibende Datei liegt in dem obigen Beispielcode in einem Unterverzeichnis «data». Diese Datei wird in ein PHP-Array zeilenweise ausgelesen. Das letzte Element des Arrays, das den Abschlusstag enthält, wird ausgelesen und anschließend aus dem Array gelöscht. Danach wird der neue Datensatz sowie der Abschlusstag, mit einem Zeilenumbruch, an das Array angehängt. Zum Schluss wird das Array wieder in die Datei geschrieben.

Sicher gibt es hier auch Methoden, mit denen z.B. nur die letzten Buchstaben der vorhergehenden Datei entfernt werden können.

GET-Methode

Diese Methode ist im Internet vor allem bei Suchmaschinen beliebt. Der gesamte Inhalt des Formulars wird über die URL in der Adressleiste des Browsers sichtbar weitergegeben. Dort steht dann unter **Aktion** z.B.

http://www.example.com/xmldata.php?Vorname=Lieschen&Nachname=M%FCller

Prinzipiell unterscheidet sich bei den XML-Formulardokumenten die Methode POST nicht von der Methode GET. In beiden Methoden wird nur ein kompletter Datensatz wie oben gezeigt als Datei weitergegeben.

Auch hier sollte, wie bei der POST-Methode, der Eintrag **Submissions → Ersetzen → Kein** gesetzt werden.

PUT-Methode

Hier wird eine lokale Adresse eingetragen. Es wird eine Datei lokal geschrieben und abgelegt. Wird die Datei nicht rechtzeitig weiter verarbeitet, so wird der Inhalt mit einem erneuten Abschicken des Formulars überschrieben. Als **Aktion** steht dort dann z.B.

`file:///home/user/Dokumente/datendatei.xml`

Auch unter Windows ist darauf zu achten, dass keine Backslashes verwandt werden. Dort also z.B.

`file:///c:/tmp/datendatei.xml`

Über ein Makro lässt sich diese Datendatei auch an eine Mail anhängen. Das Mailprogramm wird dabei gestartet und das Absenden muss nur noch veranlasst werden:

```
SUB Mailversand
    DIM attachs(0)
    IF GetGuiType() = 1 THEN
        oMailer = createUnoService("com.sun.star.system.SimpleSystemMail")
        ' Sonst Linux/Mac
    ELSE
        oMailer = createUnoService("com.sun.star.system.SimpleCommandMail")
    END IF
    oMailProgram = oMailer.querySimpleMailClient()
    oMessage = oMailProgramm.createSimpleMailMessage()
    oMessage.setRecipient("lieschen.mueller@example.org")
    oMessage.setSubject("XML-Daten")
    attachs(0) = "file:///home/user/Dokumente/datendatei.xml"
    oMessage.setAttachement(attachs())
    oMailProgram.sendSimpleMailMessage(oMessage, 0 )
END SUB
```

Komplexere Formulardefinition[5]

Das Einstiegsbeispiel hatte erst einmal nur zum Ziel, die prinzipielle Funktion eines XML-Formulars zu zeigen und dabei direkt aufkommende Probleme zu lösen. Jetzt soll es darum gehen, XML-Formulare genauer zu durchschauen.

XML-Dateien starten mit der Nennung der XML-Version und der entsprechenden Kodierung für die Schriftzeichen, die verwendet werden. Der entsprechende **`Tag`** dazu nimmt immer die erste Zeile der Datei ein.

Der gesamte weitere Inhalt wird durch **`Elemente`** strukturiert. Dabei werden für **`wohlgeformte`** XML-Dateien folgende Punkte definiert[6]:

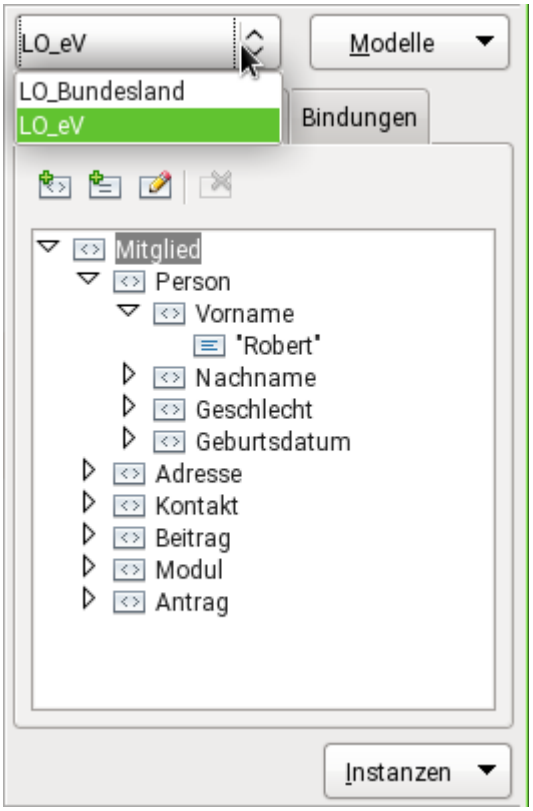
- Die XML hat genau ein **`Wurzelelement`**. Das äußere Element heißt in der Defaulteinstellung des Formulareditors <instanceData>.
- Alle Elemente haben einen Start- und einen Ende-Auszeichner. Im oberen Beispiel z.B. <Vorname>Lieschen</Vorname>. Nur wenn die Elemente keinen Inhalt einrahmen kann auch einfach ein Element mit abschließendem Bezeichner stehen: <ElementOhneInhalt />. Die Groß- und Kleinschreibung ist Teil der Bezeichnung.

5 Siehe hierzu das Beispiel «Formular_komplex.odt»
6 Siehe: *https://de.wikipedia.org/wiki/Extensible_Markup_Language*

- Ein Element der gleichen Ebene muss geschlossen werden, bevor ein neues Element geöffnet wird. Das Elternelement wird dann geschlossen, wenn alle Kindelemente geschlossen sind.
- Einem Element können verschiedene Eigenschaftszuweisungen (Attribute) zugeordnet werden. Allerdings dürften diese Attribute nicht gleiche Namen haben, da sonst die Zuweisung unklar bleibt.
- Den Attributen werden Attributeigenschaften zugeordnet, die in Anführungszeichen stehen müssen.

Definition der Instanzen

Jedes Modell hat genau eine Instanz. Werden mehrere Instanzen, z.B. für die Erstellung von Listenfeldern, benötigt, so muss über **Modelle → Hinzufügen** ein neues Modell erstellt werden. In einem neuen Modell sind Instanzen, Submissions und Bindungen neu. In einem Formular können mehrere Modelle existieren. Es kann von Modell zu Modell zum Bearbeiten umgeschaltet werden.



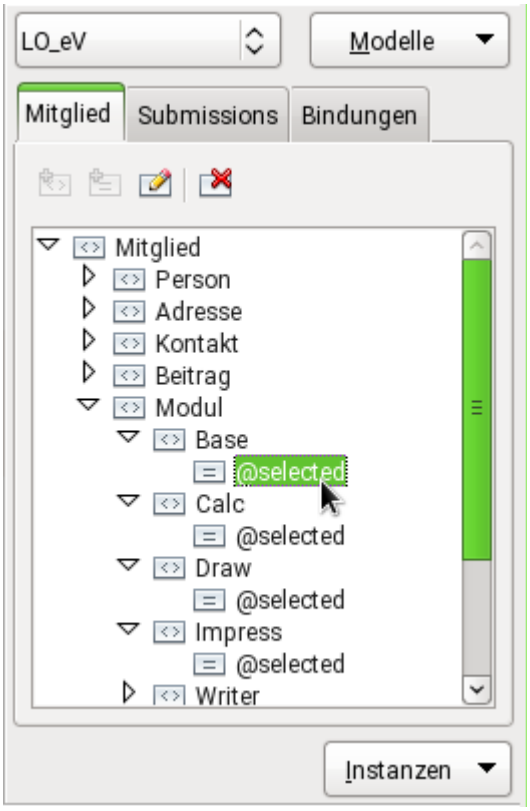
Das obige Modell «LO_eV» hat eine Instanz «Mitglied». Standardmäßig ist das Root-Element «instanceData». Die Umbenennung des Root-Elements ist zur Zeit leider über die GUI nicht möglich.

Tipp	«instanceData» ist, entgegen anderer Anleitungen zu OpenOffice 2.0, der feste Startbegriff für die Daten und kann nicht geändert werden. Eine Umbenennung ist nur über das Einlesen einer XML-Datei möglich: <?xml version="1.0" encoding="UTF-8"?> <Mitglied> </Mitglied> Wird der obige Inhalt als einfache Textdatei abgespeichert und mit dem Namen «Start.xml» versehen abgespeichert, so kann über **Instanzen → Bearbeiten** der Link zu dieser Datei eingelesen werden. «Mitglied» wird dann zum Startbegriff. Nachdem «Mitglied» als Startbegriff erscheint sollte der Link wieder aus dem Dialog entfernt werden. Ein Hinzufügen von Elementen ist sonst nur scheinbar möglich und verschwindet beim nächsten Formularstart.

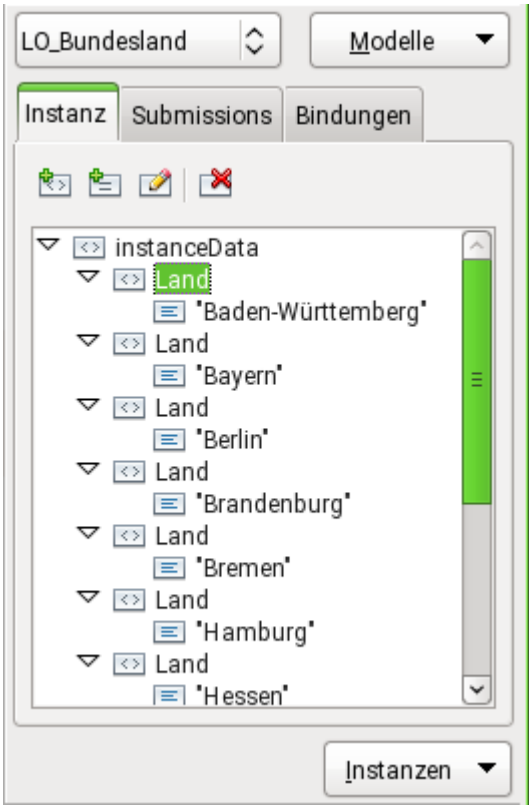
Dem Mitglied sind verschiedene Elemente baumartig untergeordnet. Enthalten Felder Werte, so sind diese über den Baum ersichtlich. Außerdem erscheinen die Werte auch im Formular selbst.

Werden den Feldern direkt auch über **Element bearbeiten → Einstellungen → Datentyp** Datentypen zugeordnet, so erstellt der Daten-Navigator automatisch die entsprechenden Bindungen, die für die Formularfelder nötig sind.

Hinweis	Datentypen können nur für Kindelemente, nicht aber für ein Elternelement eindeutig festgelegt werden. So müsste beim Element «Person» der Datentyp leer bleiben, da die Kindelemente «Vorname», «Nachname», «Geschlecht» und «Geburtsdatum» keinen einheitlichen Datentyp haben.



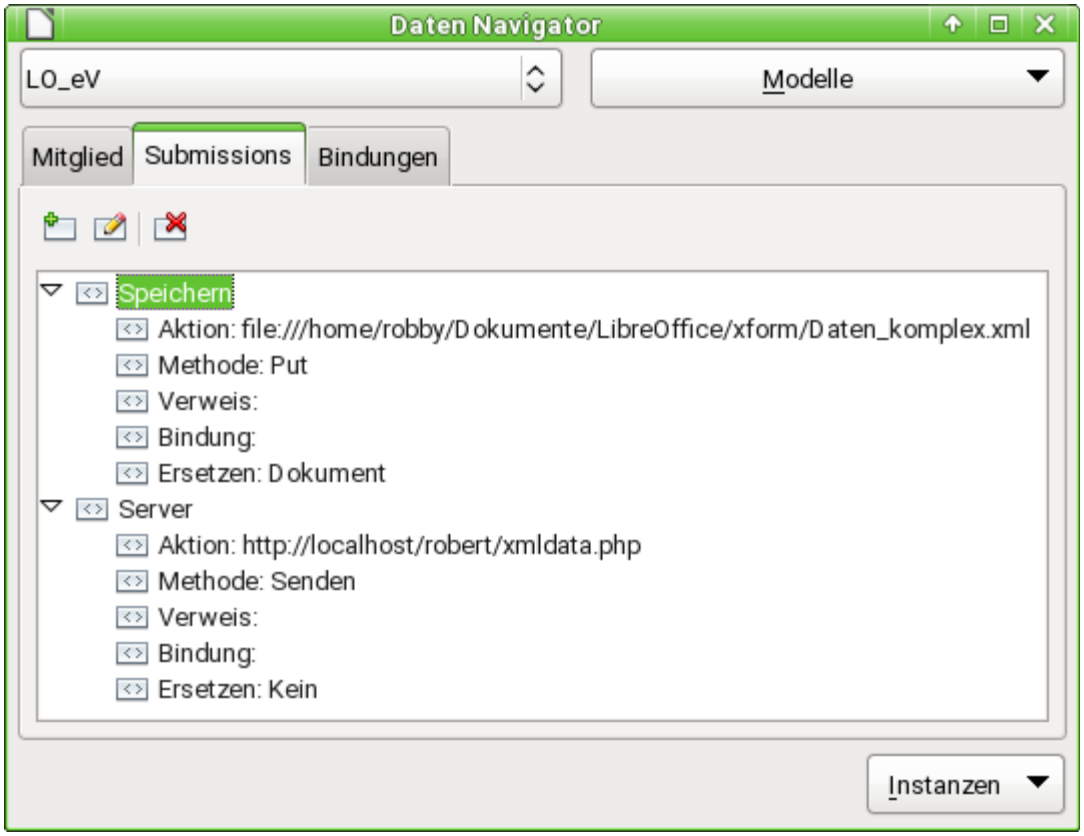
Bei den Modulen werden die Werte über Attribute, hier «selected» gespeichert. Im Daten-Navigator sind diese Werte nicht sichtbar. Die Bezeichnung mit «@» weist übrigens innerhalb von XML darauf hin, dass es sich um ein Attribut des jeweiligen Elementes handelt. Der eigentlich eingegebene Name des Attributs wird ohne «@» geschrieben.



Für den Inhalt von Listenfeldern müssen die Werte in einem neuen Modell, hier «LO_Bundesland», erstellt werden. Dabei sind alle Elemente der 2. Stufe mit gleichem Namen benannt. So kann später über die Bindungen eine Liste aller Felder abgefragt werden. Die entsprechenden Landesbezeichnungen werden in den Eigenschaften als Defaultwerte eingetragen.

Hinweis	Bei der Änderung von Eigenschaften bei den **`Instanzen`** oder auch anderen Elementen kann es passieren, dass der Daten-Navigator die Änderungen in anderen Elementen nicht aktualisiert. So macht sich z.B. eine Änderung bei den Eigenschaften eines Feldes in den Instanzen erst nach dem Abspeichern, Schließen und erneutem Öffnen des Formulardokumentes auch bei den **`Bindungen`** bemerkbar.

Daten verschicken

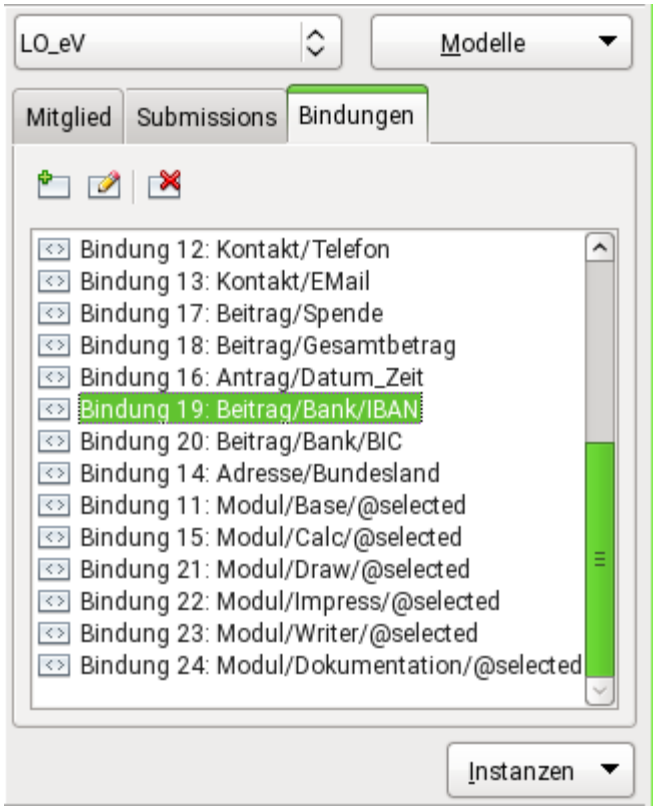


Für die Buttons zur Weiterverarbeitung werden zwei Möglichkeiten erstellt:

1. Die Möglichkeit, die Daten direkt zu speichern, kombiniert eventuell mit dem weiter oben vorgestellten Makro, die Daten in einer zentralen XML-Datei weiter zu verarbeiten.
2. Die Möglichkeit, Daten an einen Server zu schicken. Dort wird die Datei in diesem Falle mit einer *.php-Datei weiter verarbeitet.

Bindungen für Formularfelder

Die Bindungen für die Formularfelder werden dann automatisch erstellt, wenn bei Erstellen der Elemente bereits ein Datentyp angegeben wurde. Ansonsten wird hier der gesamte Pfad ohne das Wurzelelement, wie aus dem Screenshot ersichtlich, angegeben.



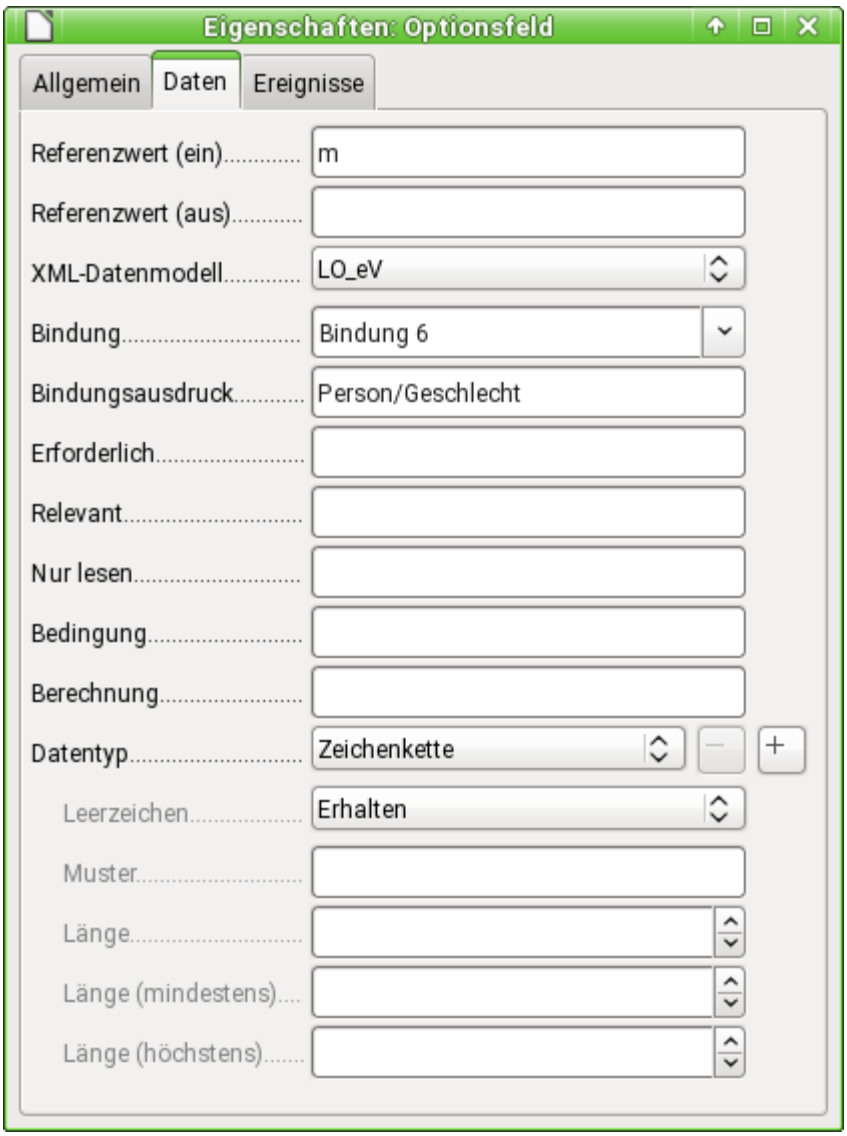
Bei der Bindungsbezeichnung muss darauf geachtet werden, dass die Bezeichnungen bei unterschiedlichen Modellen nicht gleichen. Die Namensgebung funktioniert nämlich nur innerhalb eines Modells einwandfrei. Bindungen werden dort einfach durchnummeriert. Wird eine Bindung gelöscht, so wird die Nummer anschließend wieder neu vergeben.

Hier wird die Bindung für das Listenfeld deutlich unterschiedlich zu den anderen Bindungen mit «Bindung_Land» benannt.

Formularfelder

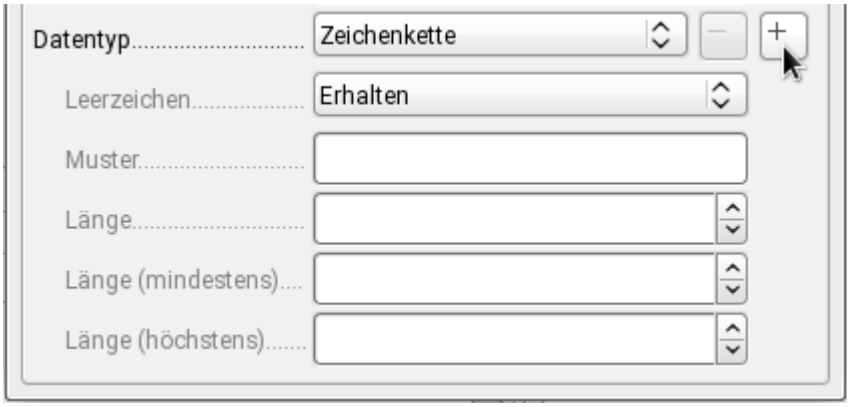
Die Eigenschaften der Formularfelder unterscheiden sich deutlich von Formularfeldern z.B. in Base. Beim Optionsfeld weist lediglich der Referenzwert den gleichen Charakter auf. Optionsfelder mit gleichem Namen zeigen auch hier die Eigenschaft, dass nur eine Option ausgewählt werden kann.

Alle weiteren Daten-Eigenschaften entsprechen den Eigenschaften, die bereits vorher für das entsprechende Feld eingestellt wurden. Zuerst wird das XML-Datenmodell «LO_eV» gewählt, dann aus diesem Modell die entsprechende Bindung. Dabei muss hier immer der Blick auf die ausgeschriebenen Bindungen des Daten-Navigators gerichtet werden. Wer weiß schon auswendig, dass sich hinter «Bindung 6» das Feld «Person/Geschlecht» verbirgt?

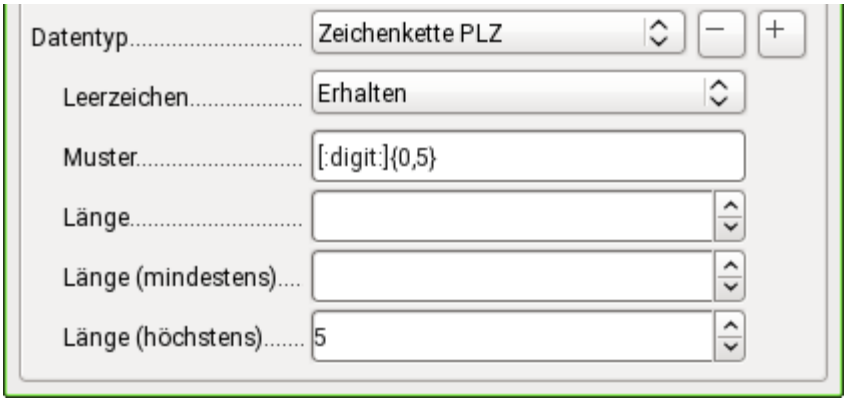


Die weiteren Einstellungen entsprechen dem der Elementdefinition. Die Eingaben unterhalb von Datentyp sind zwar ausgegraut, aber einstellbar. Hier ist allerdings das folgende Vorgehen zu beachten:

- Es sollte über **Datentyp → Zeichenkette → +** ein neuer Datentyp gewählt werden. Alle folgenden Einstellungen beziehen sich auf alle gleichlautenden Datentypen.
- Für das Muster muss mit regulären Ausdrücken (siehe LO-Hilfe) gearbeitet werden.



Ein neuer Datentyp wird erstellt, damit die Postleitzahl korrekt ist.



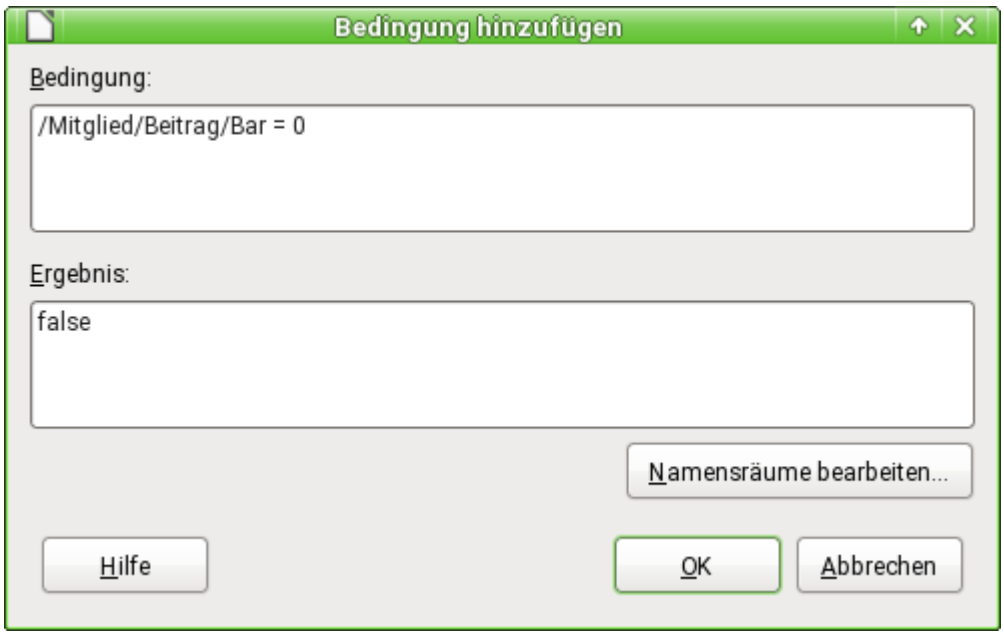
Die Elemente für den Datentyp «Zeichenkette PLZ» sind jetzt nicht mehr ausgegraut. Es ist jetzt ein Muster eingestellt, das grundsätzlich nur Zahlen zulässt. Es sollen höchstens insgesamt 5 Zahlen sein. Dies ist sowohl im Muster als auch im Längeneintrag eingestellt. Da das Feld auch leer bleiben darf, sofern keine Adresse angegeben wird, muss die Angabe der Mindestlänge und der Länge unterbleiben.

Bei den Listenfeldern wird unter **Daten → Listeneinträge aus** das Modell mit der entsprechenden Bindung ausgesucht. Die jeweiligen Werte sind übrigens direkt unter Allgemein → Listeneinträge zu sehen. Das Listenfeld wird dann aber, wie die anderen Felder, an das XML-Datenmodell LO_eV gebunden.

Eingabebedingungen

Erforderlich

Dass eine Eingabe erforderlich sein soll lässt sich einfach unter Element **Bearbeiten → Einstellungen → Erforderlich** auswählen. Dann steht unter **Bedingung** «true()» und als **Ergebnis** «true» oder «false», je nachdem, ob die Bedingung erfüllt ist oder nicht.

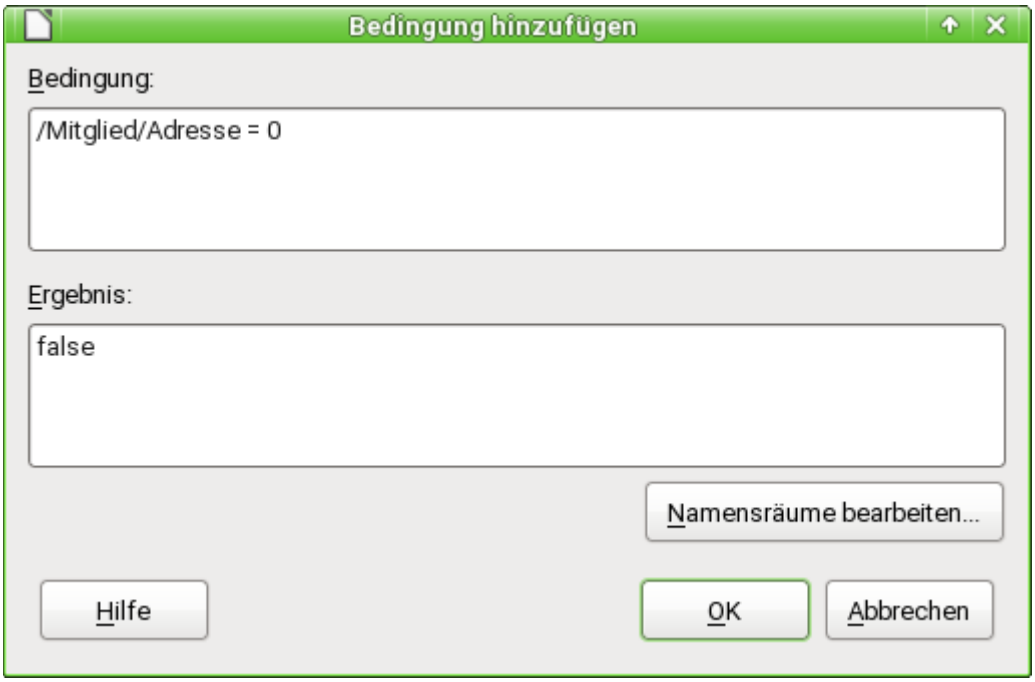


Hier wird eine Bedingung für das Feld «/Mitglied/Beitrag/Bank/IBAN» oder auch «/Mitglied/Beitrag/Bank/BIC» formuliert. Ist über das Feld «/Mitglied/Beitrag/Bar» keine Wahl erfolgt (Wert: 0), dann soll eine Eingabe erforderlich sein. Ansonsten hat sich die Eingabe in diesem Feld erledigt. Im Ergebnis weist diese Bedingung gerade darauf hin, dass «Bar» bereits gewählt wurde.

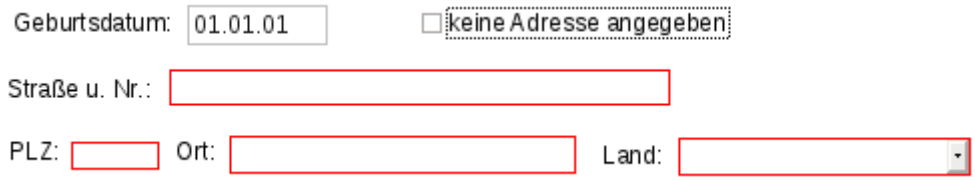
Die Bedingung ist in diesem Fall direkt von dem ersten Element aus definiert, funktioniert aber genauso gut vom Element «Beitrag» aus, da ja das erste Element nur genau einmal vorkommen darf.

Relevant

Manchmal gibt es Datenfelder, deren Eintrag andere Datenfelder beeinflusst. In der Beispieldatenbank ist dies ein Feld, das angeklickt werden kann, wenn kein Adresseintrag erfolgen soll. In den betreffenden Datenfeldern für Straße, Postleitzahl, Ort und Land werden die folgenden Ergänzungen im Bereich **Relevant** (und gleichzeitig im Bereich **Erforderlich**) getätigt:



Ist das Feld «Adresse» nicht angeklickt, dann sind die Felder «Strasse», «PLZ» usw. **`relevant`**. Das Ergebnis ist **`true`**. Eine Eingabe kann erfolgen, ist sogar **`erforderlich`**, da gleichzeitig **Erforderlich** mit der gleichen Bedingung gewählt wurde.



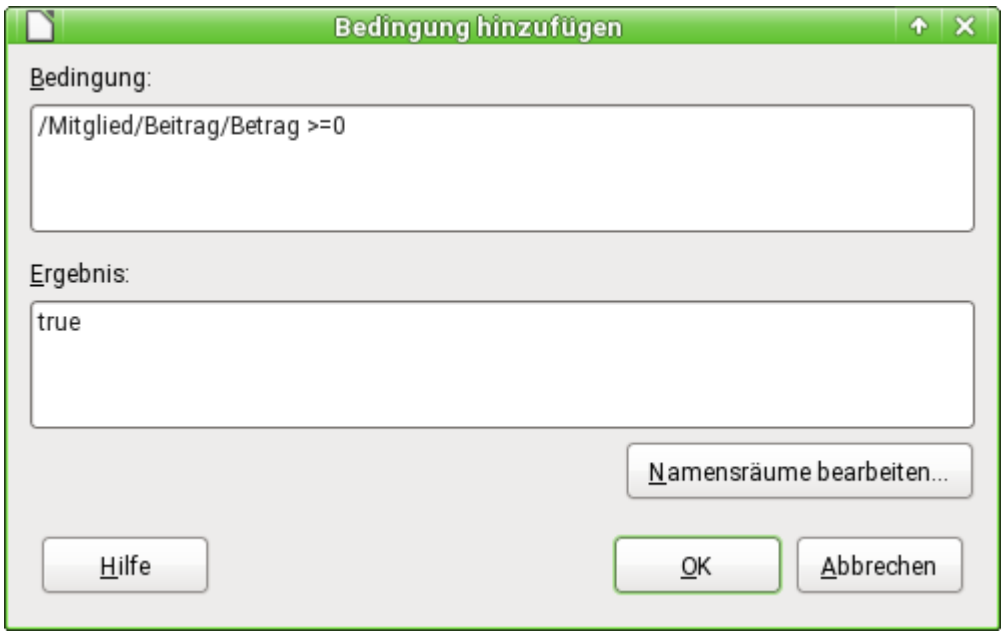
«keine Adresse angeben» ist nicht ausgewählt. Alle Felder für die Adresse erscheinen rot umrandet, da jetzt überall eine Eingabe erforderlich ist.

Wird jetzt «keine Adresse angegeben» gewählt, so werden die Felder für die Adressangabe deaktiviert. Sollen auch die Beschriftungen deaktiviert werden, so ist vorher bei den Formularfeldern die folgende Einstellung erforderlich:

In den Eigenschaften des Textfeldes, in diesem Beispiel des Formularfeldes für die Postleitzahl, ist unter **Allgemein → Beschriftungsfeld** das Beschriftungsfeld angewählt worden. Nur wenn diese Verbindung klar ist kann auch das Beschriftungsfeld deaktiviert werden.

Einschränkung

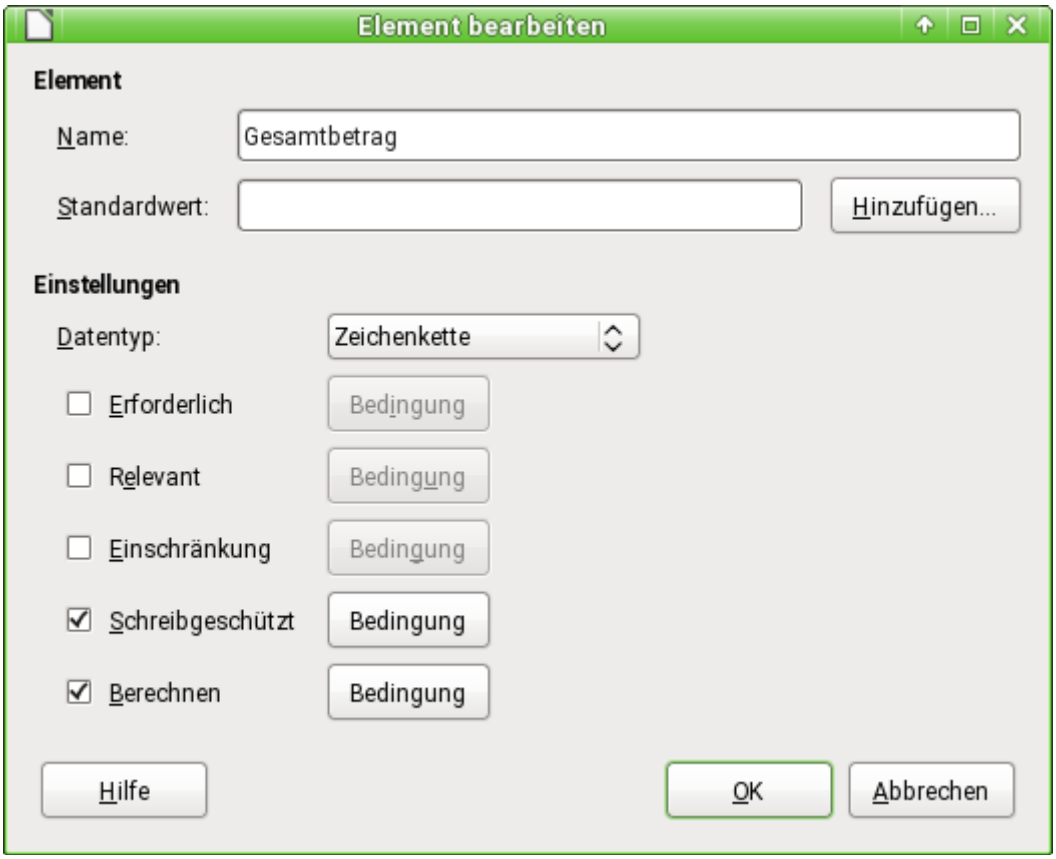
Beim «Betrag» und bei der «Spende» ist eine Einschränkung hinzugefügt worden:



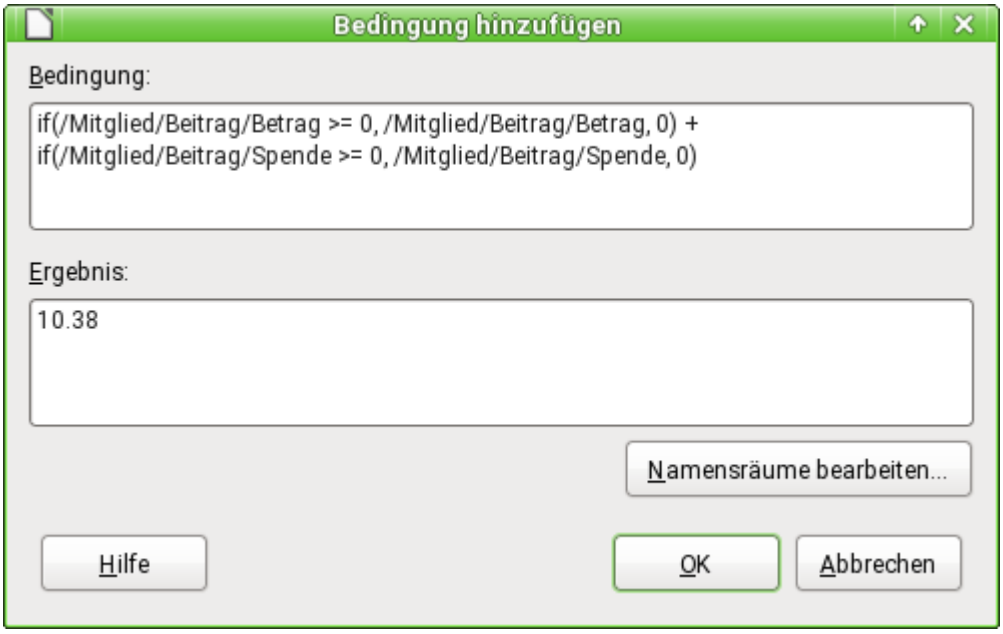
Es sollen schlicht nur Beträge erlaubt sein, die größer oder gleich 0 sind. Negative Zahlen werden also ausgeschlossen.

Schreibschutz und Berechnen

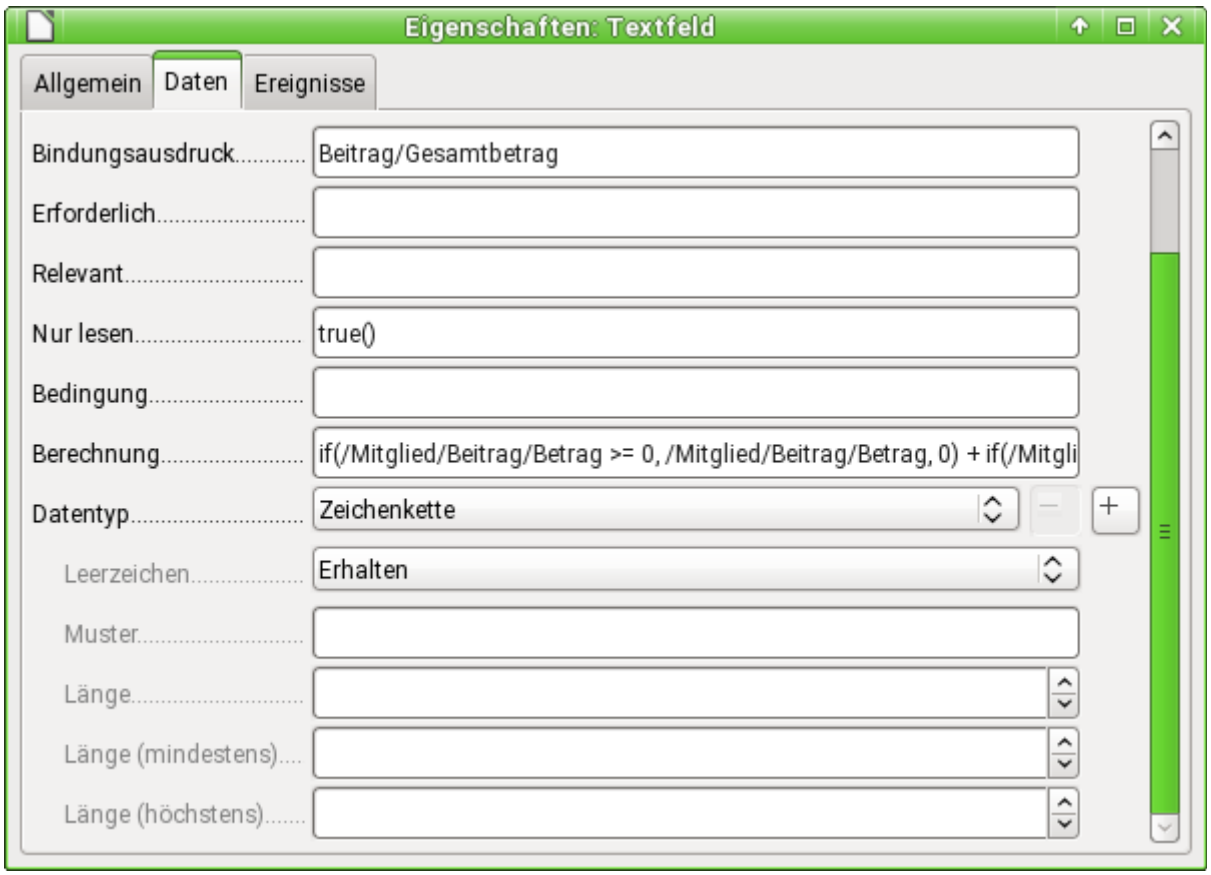
Für den Gesamtbetrag erfolgt eine Berechnung, die aber zur Zeit leider mit einem Bug behaftet ist: **`Dezimalzahlen`** werden leider intern irgendwann von Zahlen mit Dezimalpunkt in Zahlen mit Dezimalkomma umgewandelt und können so nicht dargestellt werden. Deshalb wird das Ergebnis in diesem Falle als **`Zeichenkette`** in dem Formularfeld aufgeführt.



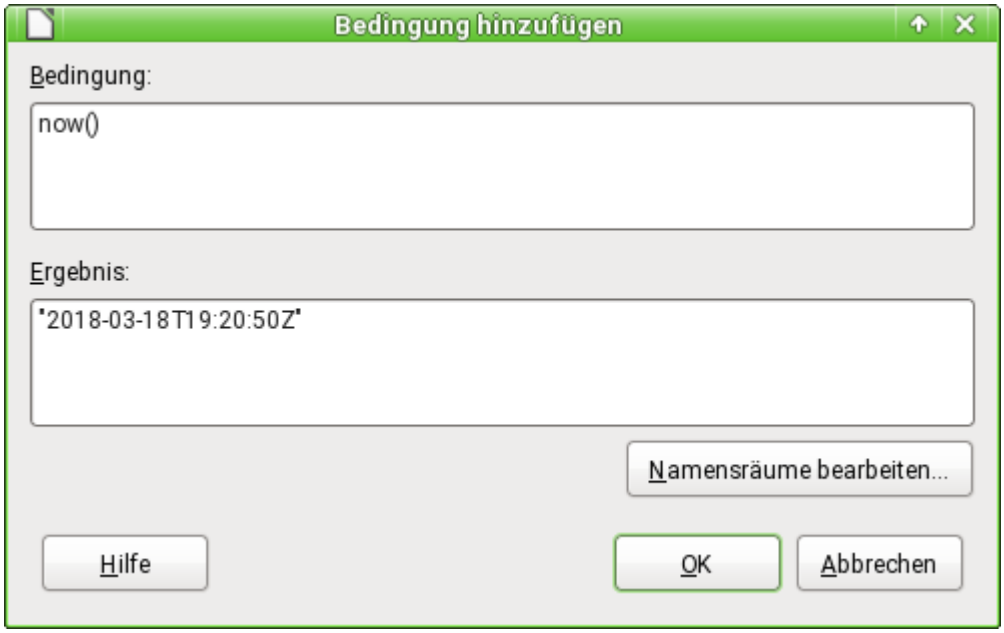
Die Berechnung erfolgt durch die Addition der Felder, die hier berücksichtigt werden sollen. Der Editor versteht hier **+**, **-**, ***** und **div** (nicht: «/», da bereits anderweitig belegt) als Rechenanweisungen. Enthält das jeweilige Feld keinen Eintrag, so muss für die Ermittlung der Summe stattdessen eine «0» angenommen werden. Sonst wird bei der Ergebnisermittlung **NaN** (not a number) ausgegeben.



In den Dateneigenschaften des Textfeldes wird dann auch diese entsprechende Berechnung anschließend sichtbar. Die Dateneigenschaften werden hier beständig aktualisiert, wenn eine Änderung in den Bedingungen der Einstellungen des Elements erfolgt.



Neben den einfachen Berechnungen stehen auch einige Funktionen zur Verfügung. Hier wurde mit der Funktion now() der aktuelle Tag und die aktuelle Zeit wieder gegeben.



Formatierte Ausgabe in XML-Dokumenten

Prinzipiell können die Daten, wie oben beschrieben, über ein Makro oder über den Server einer gemeinsamen XML-Datei zugeführt werden. Für die einfache formatierte Ausgabe wurde eine entsprechende XSL-Datei bereits vorgestellt. Bei dem jetzt anfallenden Datenanteil ist natürlich eine einfache Tabellenzeile nicht mehr ausreichend für die Daten. Hier nur ein Ausschnitt der neuen XSL-Datei, die zeigt sich auf die letzten beiden Tabellenzeilen der Darstellung im Browser konzentriert und zusätzliche Funktionen birgt:

```
<tr>
    <th></th>
    <th>Betrag</th>
    <th>Spende</th>
    <th>Gesamt</th>
    <th>Bar</th>
    <th>IBAN</th>
    <th>BIC</th>
    </tr>
<tr>
    <td></td>
    <td><xsl:value-of select="Beitrag/Betrag"/></td>
    <td><xsl:value-of select="Beitrag/Spende"/></td>
    <td><xsl:value-of select="Beitrag/Gesamtbetrag"/></td>
    <td><xsl:if test="Beitrag/Bar=1">Ja</xsl:if>
        <xsl:if test="Beitrag/Bar=0">Nein</xsl:if></td>
    <td><xsl:if test="Beitrag/Bank/IBAN=''"> -</xsl:if>
        <xsl:value-of select="Beitrag/Bank/IBAN"/></td>
    <td><xsl:if test="Beitrag/Bank/BIC=''"> -</xsl:if>
        <xsl:value-of select="Beitrag/Bank/BIC"/></td>
</tr>
```

Die Tabellenheader werden jetzt mitgeführt, so dass jedes einzelne Element auch seine Bezeichnung hat. Die Schriftart wird in den Formatierungsanweisungen für die Header verkleinert.

Wenn im Feld «Bar» eine '1' steht, dann wird 'Ja' ausgegeben, bei '0' 'Nein'. Für die IBAN bzw. BIC wird bei leeren Feldern ein ' -' eingefügt.

```
<tr>
    <th></th>
```

```
        <th colspan="5">Module</th>
        <th>Antragszeitstempel</th>
    </tr>
    <tr>
        <td></td>
        <td colspan="5">
            <xsl:if test="Modul/Base/@selected='true'">Base </xsl:if>
            <xsl:if test="Modul/Calc/@selected='true'">Calc </xsl:if>
            <xsl:if test="Modul/Draw/@selected='true'">Draw </xsl:if>
            <xsl:if test="Modul/Impress/@selected='true'">Impress </xsl:if>
            <xsl:if test="Modul/Writer/@selected='true'">Writer </xsl:if>
            <xsl:if test="Modul/Dokumentation/@selected='true'">Dokumentation </xsl:if>
        </td>
        <td><xsl:value-of select="Antrag/Datum_Zeit"/></td>
    </tr>
```

Die Module werden in einer Tabellenspalte zusammengefasst und so geschrieben, dass zwischen den Modulen jeweils ein Leerzeichen auftaucht. Die Eigenschaft des Attributes wird hierzu ausgelesen.

XML-Formulare mit Datenbankanbindung nutzen[7]

XML-Formulare speichern erst einmal nur den aktuellen Formularinhalt und geben diesen auch als xml-Datei weiter. Sie lassen sich nicht zum Navigieren durch vorherige Datenbankinhalte nutzen.

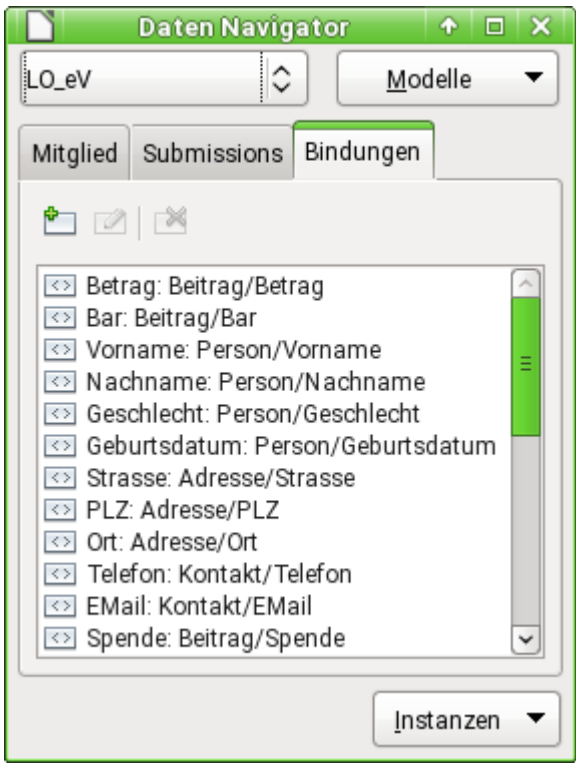
Mit Base besteht die Möglichkeit, Datensätze auszulesen und diese dann in den XML-Formularen darzustellen, zu verändern und neue Daten hinzuzufügen. Hierzu ist allerdings der Einsatz von Makros unabdingbar.

Vorteil dieser Konstruktion ist, dass Formulare so gestaltet werden können, dass dem Nutzer der Datenbankunterbau im Detail verborgen bleibt. Das Formular selbst weist hier die Datenbankverbindung nicht aus. Sie wird über Makros zur Verfügung gestellt.

Makrozugriff über die Bindungen des XML-Formulars

Der Kontakt zu den Datenfeldern des XML-Formulars erfolgt über die Bindungen des Formulars. Dazu werden zuerst einmal die Bezeichner für die Bindungen von den standardmäßigen «Bindung 1» usw. umbenannt in Bezeichner, die genau die gleiche Bezeichnung wie die Datenfelder haben:

7 Siehe hierzu das Beispiel «Formular_komplex_Base.odt» sowie die Datenbank «XML_Daten.odb»



Die Bindungen haben jetzt Bezeichner wie «Betrag», «Bar» usw. Über den folgenden Code können jetzt einzelne Datenfelder und die dazugehörigen Formularfelder mit Inhalten versehen werden.

```
oBinding = thisComponent.XForms.getByName("LO_eV").Bindings
oBinding.getByName("Vorname").setValue("Otto")
```

Das Writer-Dokument («thisComponent») bietet den Kontakt zu den Bindungen an. Die einzelne Bindung wird mit dem Namen ausgewählt und der Wert in die Bindung geschrieben. Der Wert erscheint dann direkt in den Elementen und in den Formularfeldern.

Mit dieser Methode lässt sich auch ein Formular zurücksetzen, so dass dort keine Werte mehr stehen. XML-Formulare bieten diese Möglichkeit sonst nicht. Allerdings fehlt den Formularfelder in XML-Formularen eine Eigenschaft, die Formularfelder mit Datenbankanbindung haben: Sie lassen sich nicht über die GUI auf **`NULL`** setzen. Dadurch wird bei einem leeren Text in Datumsfeldern stattdessen '01.01.1990' ausgegeben. Bei numerischen Feldern erscheint, abhängig von der Formatierung, z.B. '0,00 €'. Das ist nicht nur lästig, wenn z.B. zuerst das Defaultdatum gelöscht wird. Es führt auch leicht dazu, dass das Defaultdatum direkt mit abgespeichert wird.

Um Datumsfelder auf leere Felder setzen zu können muss auf den Controller zugegriffen werden:

```
oForm = ThisComponent.Drawpage.Forms.getByIndex(0)
oController = ThisComponent.getCurrentController()
oFieldGeburtsdatum = oForm.getByName("datGeburtsdatum")
oViewGeb = oController.getControl(oFieldGeburtsdatum)
oViewGeb.setEmpty
```

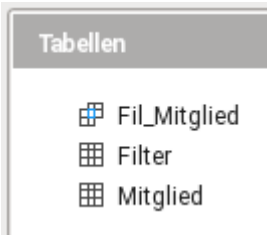
Bei numerischen Feldern gibt es die Methode **`setEmpty`** nicht. Hier werden die Felder mit **`setText`** zurückgesetzt:

```
oFieldBetrag = oForm.getByName("numBetrag")
oViewBet = oController.getControl(oFieldBetrag)
oViewBet.setText("")
```

Jetzt kann das Formular mit Daten gefüllt und für die Eingabe von neuen Daten zurückgesetzt werden.

Base-Tabellen erstellen

Für die Datenbank werden 2 Tabellen und eine Ansicht erstellt:



Die Tabelle "Mitglied" enthält alle Daten, die abgespeichert werden sollen. Zusätzlich zu den ursprünglich im XML-Formular enthaltenen Feldern wird ein Feld "ID" für die Speicherung des automatisch erstellten Primärschlüssels erstellt. Dieses Feld muss auch in das XML-Formular als Element übernommen werden, braucht aber nicht als Formularfeld zu erscheinen.

Die Tabelle "Filter" wird dazu benutzt, über das Formular die Datensätze nach bestimmten "Nachnamen" zu durchsuchen. Deshalb enthält diese Tabelle lediglich ein Feld "ID" als Ja/Nein-Feld, das gleichzeitig der Primärschlüssel ist, und ein Feld "Nachname". Die Tabelle besteht grundsätzlich nur aus einem Datensatz, der immer wieder überschrieben wird.

In der Ansicht "Fil_Mitglied" sind alle Datensätze aufgeführt, die der Filterung aus der Tabelle "Filter" entsprechen:

```
SELECT * FROM "Mitglied" WHERE
   IFNULL( LOWER ( "Nachname" ), '' ) LIKE '%' ||
   IFNULL( ( SELECT LOWER ( "Nachname" ) FROM "Filter" WHERE "ID" =
   TRUE ), '' ) || '%'
```

Aus der Tabelle "Filter" wird der "Nachname" für den Datensatz ausgelesen, bei dem "ID" auf **Wahr** (**TRUE**) gesetzt ist. Die Eingabe wird in Kleinbuchstaben umgewandelt, damit der Vergleich nicht an den Unterschieden zwischen Groß- und Kleinschreibweise scheitert. Der ausgelesen Inhalt wird mit dem Inhalt des Feldes "Nachname", ebenfalls in Kleinschreibweise umgewandelt, über die Funktion **LIKE** verglichen. Die Position des Filterwertes ist dabei so gesetzt, dass beliebig viele Zeichen vor und nach dem Filterwert vorkommen können. So gibt die Abfrage alle Datensätze wieder, bei denen z.B. der "Nachname" an irgendeiner Stelle den Buchstaben 'k' enthält, wenn dieser als Filterwert eingeben wurde.

Damit die Möglichkeit besteht, auch alle Datensätze über den Filter anzeigen zu können, muss bei einer leeren Eingabe nachgeholfen werden. Die leere Eingabe wird über **IFNULL** mit einem leeren Text ersetzt. Gleiches gilt auch für eventuell leer gebliebene Eingaben in das Feld "Nachname" in der Tabelle "Mitglied". Diese Datensätze würden sonst nicht mehr auftauchen.[8]

Makrozugriff auf die Base-Datenbank

Der Zugriff zu Datenbanken ist im Detail im Base-Handbuch beschrieben. Hier deshalb nur einige Kommentare zu wichtigen Details.

Die erste Prozedur DatenTabelle_Start muss beim Öffnen des Formulars direkt gestartet werden. Sie wird deshalb über **Extras → Anpassen → Ereignisse → Ansicht wurde erzeugt** in das Formular eingebunden.

```
SUB DatenTabelle_Start
   oXForm = ThisComponent
   stDir = Left(oXForm.Location,Len(oXForm.Location)-Len(oXForm.Title))
   stDir = ConvertToUrl(stDir & "XML_Daten.odb")
   oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
   oDatasource = oDatabaseContext.getByName(stDir)
```

8 Siehe dazu im Detail das Base-Handbuch, besonders die Kapitel Abfragen und Datenfilterung

```
oConnection = oDatasource.GetConnection("","")
oStatement = oConnection.createStatement()
oStatement.ResultSetType = com.sun.star.sdbc.ResultSetType.SCROLL_SENSITIVE
```

Der **ResultSetType** muss zum freien Scrollen freigeschaltet werden. Ansonsten kann nur vorwärts durch die Datensätze gescrollt werden. Deswegen hier der Zusatz **SCROLL_SENSITIVE**. So könnten auch Datenänderungen in den bestehenden Datensätzen nachvollzogen werden. Mit der internen HSQLDB ist es allerdings nicht möglich, dem **ResultSet** auch neue Daten hinzuzufügen. Daher kann hier ohne Probleme auf die nicht schreibbare Ansicht "Fil_Mitglied" zugegriffen werden. Für Datenänderungen wird später auf den direkten SQL-Code zurückgegriffen.

```
stQuery = "SELECT ""ID"", ""Vorname"", ... , ""Datum_Zeit"" FROM ""Fil_Mitglied"""
oResult = oStatement.executeQuery(stQuery)
IF oResult.last THEN
   stLast = oResult.getRow
ELSE
   stLast = "0"
END IF
```

Die Abfrage ist hier nur verkürzt dargestellt. Sie umfasst alle Felder der Tabelle und könnte auch über

```
SELECT * FROM "Fil_Mitglied"
```

erstellt werden. Da im Code aber der Bezug zu den einzelnen Feldern klarer sein sollte wird hier jedes Feld einzeln benannt.

Nach dem Erstellen des Ergebnisobjektes **oResult** wird zuerst zum letzten Datensatz gescrollt und dort die Nummer des Datensatzes über **getRow** abgefragt. Anschließend wird wieder zum ersten Datensatz gescrollt und die Prozedur **WerteInFormular** aufgerufen, die die Werte aus der Datenbank in das Formular überträgt. Ist noch kein Datensatz vorhanden, so wird stattdessen das Formular zurückgesetzt.

```
IF oResult.first THEN
   stPosition =  oResult.getRow
   WerteInFormular
ELSE
   stPosition = "0"
   WerteInFormularReset
END IF
```

Um das Filterfeld in dem Formular noch mit dem entsprechenden Inhalt zu versorgen wird auch noch die Tabelle "Filter" kurz abgefragt. Der Inhalt wird direkt dem Filterfeld zugewiesen. Das Filterfeld ist ansonsten nicht Element des XML-Formulars.

```
   oStatementF = oConnection.createStatement()
   stSql = "SELECT ""Nachname"" FROM ""Filter"" WHERE ""ID"" = TRUE"
   oResultF = oStatementF.executeQuery(stSql)
   oResultF.Next
   stFilter = oResultF.getString(1)
   ThisComponent.Drawpage.Forms.getByIndex(0).
      getByName("Textfeld_Filter_Nachname").Text = stFilter
END SUB
```

Für die Navigation müssen einige Variablen dieser Prozedur auch außerhalb der Prozedur weiter verfügbar sein. Sie werden zu Beginn des Moduls notiert. Sobald allerdings über Extras → Makros → Makros bearbeiten der Makroeditor geöffnet wird, sind die in den globalen Variablen gespeicherten Inhalte gelöscht. Das führt dann dazu, dass das Scrollen durch die Datenbank nicht mehr möglich ist und das Formular stattdessen neu gestartet werden muss.

```
GLOBAL oConnection AS OBJECT
GLOBAL oStatement AS OBJECT
GLOBAL oResult AS OBJECT
GLOBAL stLast AS STRING
GLOBAL stPosition AS STRING
```

Die Naviationsleiste im Formular soll so aussehen:

Der Wert für **`stPosition`** ist in der Abbildung '1', der Wert für **`stLast`** '4'. Jetzt sind Navigationsbuttons mit Prozeduren zu verbinden.

Mit **`oResult.First`** wird der erste Datensatz angesprungen und die entsprechenden Werte für diesen Datensatz werden über **`WerteInFormular`** in das Formular eingelesen.

```
SUB ErsterDatensatz
    IF oResult.First THEN
        stPosition = oResult.getRow
    ELSE
        stPosition = "0"
    END IF
    WerteInFormular
END SUB
```

Das Scrollen zum vorhergehenden Datensatz ist etwas komplizierter, da nicht vor den ersten Datensatz gesprungen werden soll. Dies wird in der ersten Bedingung abgefangen. Die zweite Bedingung klärt den Fall, dass vorher ein neuer Datensatz angesprungen wurde und die Position des neuen Datensatzes größer war als die des letzten existierenden Datensatzes. Unter dieser Bedingung soll zum vorher letzten Datensatz gesprungen werden. Ansonsten ist einfach über **`oResult.previous`** der Sprung zum vorherigen Datensatz mit Abfrage der Position möglich.

```
SUB VorherigerDatensatz
    IF oResult.isFirst THEN
        stPosition = oResult.getRow
    ELSEIF stPosition > stLast THEN
        oResult.absolute(stLast)
        stPosition = stLast
    ELSE
        oResult.previous
        stPosition = oResult.getRow
    END IF
    WerteInFormular
END SUB
```

Zum nächsten Datensatz geht es mit **`oResult.next`**. Ist bereits der letzte Datensatz erreicht, so wird die Position um eine Position höher als der eigentlichen Position angesetzt. Das Formular wird außerdem über **`WerteInFormularReset`** von allen Inhalten geleert. Ähnlich wird auch beim letzten Datensatz verfahren.

```
SUB NaechsterDatensatz
    IF oResult.isLast THEN
        stPosition = oResult.getRow + 1
        WerteInFormularReset
    ELSE
        oResult.next
        stPosition = oResult.getRow
        WerteInFormular
    END IF
END SUB

SUB LetzterDatensatz
    IF oResult.isLast THEN
        stPosition = oResult.getRow + 1
        WerteInFormularReset
    ELSE
        oResult.Last
        stPosition = oResult.getRow
        WerteInFormular
    END IF
END SUB
```

Der neue Datensatz wird über den letzten Datensatz erreicht. Anschließend wird einfach die Position so angezeigt, dass z.B. '4' von '3' Datensätzen angezeigt würden. Das Formular wird wieder geleert.

```
SUB NeuerDatensatz
    oResult.Last
    stPosition = oResult.getRow + 1
    WerteInFormularReset
END SUB
```

Für das Speichern werden die Felder über die Bindungen augelesen. Dabei muss der Datentyp für die Funktion **`getValue()`** angegeben werden. Hier wird grundsätzlich der Datentyp **`"sting"`** gewählt.

```
oBinding = thisComponent.XForms.getByName("LO_eV").Bindings
stID = oBinding.getByName("ID").getValue("string")
stVorname = String_to_SQL(oBinding.getByName("Vorname").getValue("string"))
...
```

Existiert bereits ein Eintrag für das Feld «ID», so handelt es sich um einen bereits existierenden Datensatz. Folglich muss über SQL ein Upadte erfolgen. Andernfalls handelt es sich um einen neuen Datensatz. Der SQL-Befehl hierfür ist der Insert-Befehl. Nur bei einem neuen Datensatz muss anschließend die Tabelle neu eingelesen werden, damit das Scrollen durch die Datensätze wieder klappt. Außerdem wird dann direkt zum letzten Datensatz navigiert.

```
IF stID <> "" THEN
    'Update
    stSql = "UPDATE ""Mitglied"" SET ""Vorname"" = "+stVorname+",
          ""Nachname"" = "+stNachname+"," ...
    stSql = stSql + " WHERE ""ID"" = "+stID+""
    oStatement1 = oConnection.createStatement()
    oStatement1.executeUpdate(stSql)
ELSE
    'Insert
    stSql = "INSERT INTO ""Mitglied"" (""Vorname"",""Nachname"",""," ... ")"
    stSql = stSql + " VALUES ("+stVorname+","+stNachname+"," ... ")"
    oStatement1 = oConnection.createStatement()
    oStatement1.executeUpdate(stSql)
    DatenTabelle_Start
    oResult.Last
    stPosition = oResult.getRow
    WerteInFormular
END IF
```

Die Werte in dem XML-Formular werden als String ausgelesen. Dabei sollen nacheinander folgende Änderungen in den Strings vorgenommen werden, damit sie von der Datenbank einwandfrei verarbeitet werden können:

1. Hochkommata (') werden in SQL als Textbegrenzer eingesetzt. Kommen diese Hochkommata allerdings innerhalb eines Textes vor, so müssen sie mit einem weiteren Hochkomma maskiert werden.
2. Ist der String leer, so soll an die Datenbank kein leerer String ('') weitergegeben werden. Stattdessen wird NULL, also «keine Daten», an die Datenbank gesandt.
3. Besteht der String aus der Zahl 1 oder 0, so ist dies ohne Hochkommata weiter zu geben. Ansonsten wird es als Text betrachtet und vor allem nicht für Ja/Nein-Felder als Alternative zu True/False angesehen.
4. Trifft weder Punkt 2 noch Punkt 3 zu, so handelt es sich um einen Ausdruck, der problemlos in Hochkommata eingeschlossen weitergegeben werden kann.

```
FUNCTION String_to_SQL(st AS STRING)
    IF InStr(st,"'") THEN
        st = Join(Split(st,"'"),"''")
    END IF
    IF st = "" THEN
        st = "NULL"
    ELSEIF st = "1" OR st = "0" THEN
```

```
        st = st
    ELSE
        st = "'" & st & "'"
    END IF
    String_to_SQL = st
END FUNCTION
```

In XML-Formularen hat die Angabe eines Zeitstempels das Format '2018-03-28T19:30:27Z'. Für die Weitergabe in SQL muss das 'T' durch eine Leerstelle ersetzt werden. Das 'Z' muss von dem Zeitstempel abgetrennt werden, so dass schließlich '2018-03-28 19:30:27' abgespeichert werden kann.

```
FUNCTION XMLTime_to_SQLTime(st AS STRING)
    st = Join(Split(st,"T")," ")
    st = Join(Split(st,"Z"),"")
    XMLTime_to_SQLTime = st
END FUNCTION
```

Für die Datenfilterung ist der Name des Textfeldes in den Zusatzinformationen des Buttons (**Tag**) abgespeichert. Hier wird er ausgelesen und darüber dann der Inhalt des Textfeldes bestimmt. Die Tablle "Filter" erhält ein Update und anschließend wird die Datentabelle neu eingelesen.

```
SUB Filtern(oEvent AS OBJECT)
    oButton = oEvent.Source.Model
    stFilter = oButton.Parent.getByName(oButton.Tag).Text
    stSql = "UPDATE ""Filter"" SET ""Nachname"" = '"+stFilter+"' WHERE ""ID"" = TRUE"
    oStatementF = oConnection.createStatement()
    oStatementF.executeUpdate(stSql)
    DatenTabelle_Start
END SUB
```

Hiermit lässt sich jetzt durch einen Datenbank scrollen, ein Datensatz bearbeiten und ein neuer Datensatze eingeben. Auch die Suche in vorhandenen Datensätzen ist möglich.

Abgespeicherte XML-Dateien einlesen[9]

Werden über das Formular Daten als XML-Datei abgespeichert, so kann es zur Auswertung auf einem anderen Rechner auch sinnvoll sein, die Daten wider einlesen zu können. Gegbenenfalls können so Daten aus vielen Formularen z.B. in eine Datenbank über das Formular eingelesen, mit anderen Daten zusammengeführt und wieder abgespeichert werden.

Eine abgespeicherte XML-Datei hat z.B. folgenbden Inhalt:

```
<?xml version="1.0" encoding="UTF-8"?>
<Mitglied><Person><Vorname>Lara</Vorname><Nachname>Schmidtke</Nachname><Geschlecht>w<
/Geschlecht><Geburtsdatum>2018-01-
01</Geburtsdatum></Person><Adresse>1</Adresse><Kontakt><Telefon>08754890123</Telefon>
<EMail>lara64@example.com</EMail></Kontakt><Beitrag><Bar>1</Bar><Betrag>0</Betrag><Sp
ende>10</Spende><Gesamtbetrag>10</Gesamtbetrag><Bank><IBAN/><BIC/></Bank></Beitrag><M
odul><Base selected="false"/><Calc selected="false"/><Draw selected="true"/><Impress
selected="false"/><Writer selected="false"/><Dokumentation
selected="false"/></Modul><Antrag><Datum_Zeit>2018-03-
27T16:47:44Z</Datum_Zeit></Antrag></Mitglied>
```

Die Daten dieser Datei sollen jetzt in das Formular übertragen werden. Wie beim Kopieren der neuen Daten in eine dauerhafte Datendatei wird hier zuerst der Inhalt der XML-Datei eingelesen.

```
SUB Import
    oDoc = ThisComponent
    stDir = Left(oDoc.Location,Len(oDoc.Location)-Len(oDoc.Title))
    stFile = stDir & "Beispiel_4.xml"
    oFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
    IF oFileAccess.exists(stFile) THEN
        oInputStream = createUnoService("com.sun.star.io.TextInputStream")
        oFile = oFileAccess.OpenFileReadWrite(stFile)
```

9 Siehe hierzu das Beispiel «Formular_komplex_Base_Import.odt»

```
        oInputStream.SetInputStream(oFile.getInputStream)
        DO WHILE NOT oInputStream.isEOF
            stTmp = stTmp & oInputStream.ReadLine()
        LOOP
        oInputStream.closeInput
```

Nach dem Einlesen wird zuerst der einführende Inhalt der XML-Datei von den eigentlichen Daten getrennt. Trennsymbol ist hier **?>**.

```
        a = Split(stTmp,"?>")
```

Die nachfolgenden Felder haben den folgenden Aufbau:

```
<Mitglied><Person><Vorname>Lara</Vorname><Nachname>Schmidtke</Nachname><Geschlecht>w<
/Geschlecht><Geburtsdatum>2018-01-01</Geburtsdatum></Person><Adresse>1</Adresse> ...
```

Wird jetzt nach **><** getrennt, so bleibt das Element und der Inhalt des Elementes grundsätzlich zusammen bestehen:

```
        aTmpData = split(a(1),"><")
```

Dies führt zu folgender Trennung:

```
<Mitglied
Person
Vorname>Lara</Vorname
Nachname>Schmidtke</Nachname
Geschlecht>w</Geschlecht>
...
Modul
Base selected="false"/>
...
```

Aus diesem Array müssen jetzt die Elemente (oder aus Datenbanksicht die Feldbezeichnungen) sowie die Inhalte, die zu dem jeweiligen Element gehören, extrahiert werden. Nur Arrayelemente mit **</** und **=** (siehe die Auswahlelemente aus den Modulen) müssen weiter bearbeitet und in das Formular übernommen werden. Sie werden nach den jeweils entsprechenden Kriterien wieder in Elemente (Feldbezeichnungen) und Daten aufgesplittet.

```
        k = 0
        FOR i = 0 TO UBound(aTmpData)
            IF InStr(aTmpData(i),"</") OR InStr(aTmpData(i),"=") THEN
                REDIM PRESERVE aField(k)
                REDIM PRESERVE aData(k)
                IF InStr(aTmpData(i),"</") THEN
                    b = split(aTmpData(i),">")
                    c = split(b(1),"</")
                    aField(k) = b(0)
                    aData(k) = c(0)
                END IF
                IF Instr(aTmpData(i),"=") THEN
                    b = split(aTmpData(i)," ")
                    c = split(b(1),"""")
                    aField(k) = b(0)
                    aData(k) = c(1)
                END IF
                k = k+1
            END IF
        NEXT i
```

Anschließend wird das gesamte Formular, wie bei der Eingabe neuer Daten, erst einmal von allem Inhalt geleert. Dann wird die Verbindung zu den Bindungen des XML-Formulars aufgenommen und die Daten der jeweiligen Bindung zugeschrieben.

```
        WerteInFormularReset
        oBinding = oDoc.XForms.getByName("LO_eV").Bindings
        FOR i = 0 TO UBound(aField)
            oBinding.getByName(aField(i)).setValue(aData(i))
        NEXT i
    END IF
END SUB
```

Das Makro funktioniert, ähnlich wie die Makros zu Datenbankanbindung, natürlich nur dann einwandfrei, wenn die Bindungen die gleichen Bezeichnungen haben wir die entsprechenden Elemente.

Anhang

Submissions-Methoden[10]

Method	*Serialization*	*Schemes*
post	xml	http(s) mailto
put	xml	http(s) file
get	url encoded	http(s) file
urlencoded-post	url encoded	http(s) mailto
form-data-post	multipart form data	http(s) mailto
multipart-post	multipart related	http(s) mailto

Xforms Funktionen[11]

Function	*Arguments*	*Returns*	*Description*
avg	node-set	number	Average
boolean-from-string	string	boolean	Type conversion
count-non-empty	node-set	number	Count non-empty
days-from-date	string	number	Days in epoch
if	boolean, string, string	string	Conditional
index	string	number	Repeat index
instance	string	node-set	Locate instance
max	node-set	number	Maximum
min	node-set	number	Minimum
months	string	number	Months in period
now	-	string	Current time
property	string	string	Feature value
seconds	string	number	Seconds in period
seconds-from-dateTime	string	number	Seconds in epoch

XPath Operators

`|`	Union
`*`	Wildcard
`[]`	Predicate
`+ - * div mod`	Arithmetic
`= < <= > >= != and or`	Boolean

10 Siehe *https://www.w3.org/MarkUp/Forms/2006/xforms-qr*
11 Siehe *https://www.w3.org/MarkUp/Forms/2006/xforms-qr*

LibreOffice

/libreoffice.org

ask.libreoffice.org

Dokumentationsteam

XML-Formulare

Erstellen Sie XML-Formulardokumente

Zu dieser Anleitung:
Bei XML-Formularen handelt es sich um Formulare, die während der Eingabe Berechnungen ermöglichen, Felder in Abhängigkeit von anderen Feldern aktivieren, deutlich auf fehlende Eingaben hinweisen usw. Über die regulären Ausdrücke von LibreOffice lassen sich die Inhalte, die in einem Formularfeld eingegeben werden dürfen, sehr eng umschreiben. Berechnungen anderer Felder während der Eingabe in ein Feld, Definition von Mindestbeträgen für Zahlenfelder, Einsatz von Funktionen in versteckten Feldern usw. sind möglich.
Der Inhalt der Formulare wird entweder in eine lokale XML-Datei geschrieben oder über das Netz an einen Server zur Weiterverarbeitung übertragen.

Zu den Autoren:
Dieses Buch entstand durch Freiwillige der LibreOffice-Gemeinschaft.

Eine PDF-Version dieses Buches und aller Updates zu diesem Buch kann frei unter http://de.libreoffice.org/get-help/documentation heruntergeladen werden.

Über LibreOffice:
LibreOffice ist eine leistungsfähige Office-Suite, für verbreitete Betriebssysteme wie Windows, GNU/Linux 32-/64-Bit und Apple Mac OS X geeignet. Es bietet sechs Anwendungen für die Erstellung von Dokumenten und zur Datenverarbeitung:
Writer, Calc, Impress, Draw, Base und Math.

LibreOffice entsteht aus der kreativen Zusammenarbeit von Entwicklern und der Gemeinschaft der Stiftung "The Document Foundation". Die Stiftung hat ihren Sitz in Deutschland.

LibreOffice kann unter der Adresse http://de.libreoffice.org/download kostenlos heruntergeladen werden.