LibreOffice
The Document Foundation

XML-Formulare

Formulardokumente erstellen und Daten auslesen

7.6

Copyright

Autoren

Robert Großkopf

Mitwirkende

Klaus-Jürgen Weghorn

Susanne Mohn

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 01.08.2023. Basierend auf der Version LibreOffice 7.6.

Inhalt

Einleitung

Mit XML-Formulardokumenten werden Formulare erstellt, die XML-Dateien erzeugen. Diese werden in ihrer entsprechenden Struktur mit Daten versehen, die sowohl in der Formulardatei als auch als *.xml-Datei 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 mit 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.

Vorteile von XML-Formularen gegenüber anderen Formularen in LibreOffice:

- Die Eingabe in ein Feld kann sehr genau durch entsprechende Typvorgaben eingegrenzt werden.
- Felder, bei denen eine Eingabe notwendig ist, sind durch eine entsprechende rote Umrandungsfarbe deutlich erkennbar.
- Felder können in Abhängigkeit von anderen Feldern als notwendig oder nicht notwendig definiert werden.
- Felder können als Berechnungsfelder anderer Felder erstellt werden.
- All dies geschieht unmittelbar während der Eingabe. Falsches Ausfüllen der Formulare kann also sehr stark eingeschränkt werden.

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

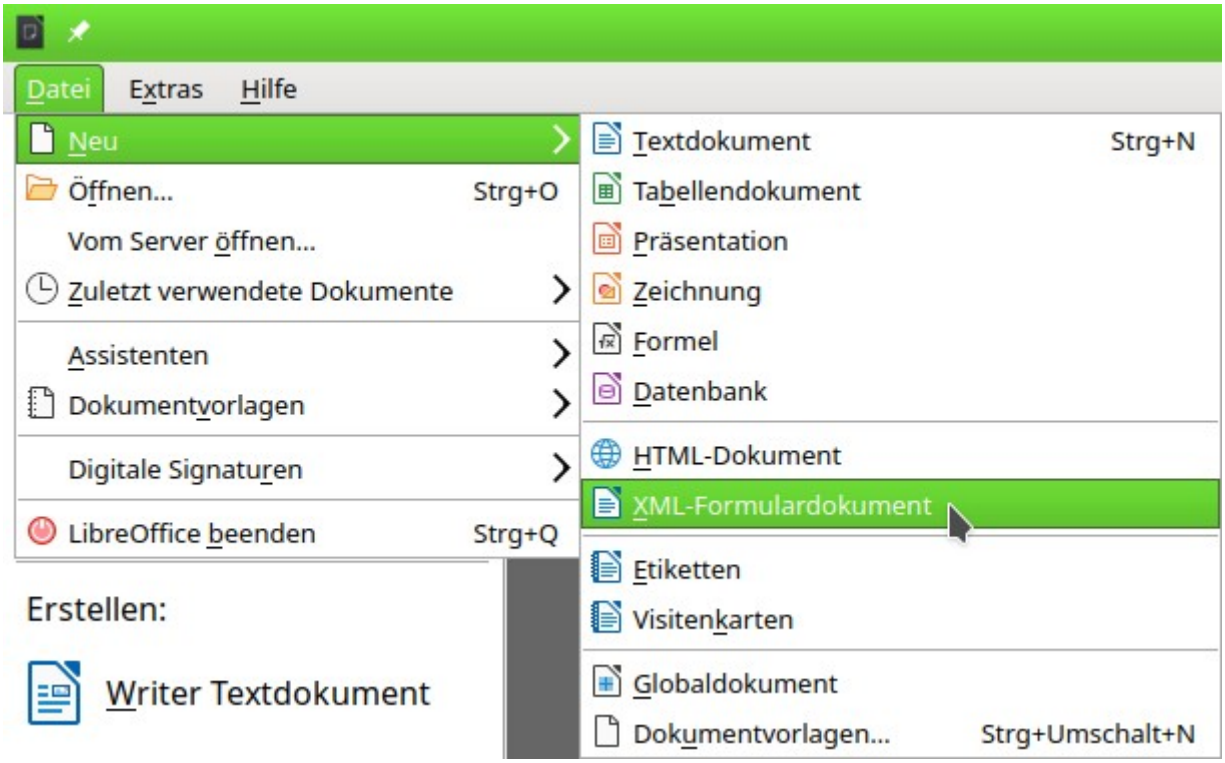
XML-Dateien, die das Formular erzeugt, haben den folgenden Aufbau:[1]

```
001 <?xml version="1.0" encoding="UTF-8"?>
002 <instanceData>
003    <Vorname>Lieschen</Vorname>
004    <Nachname>Müller</Nachname>
005 </instanceData>
```

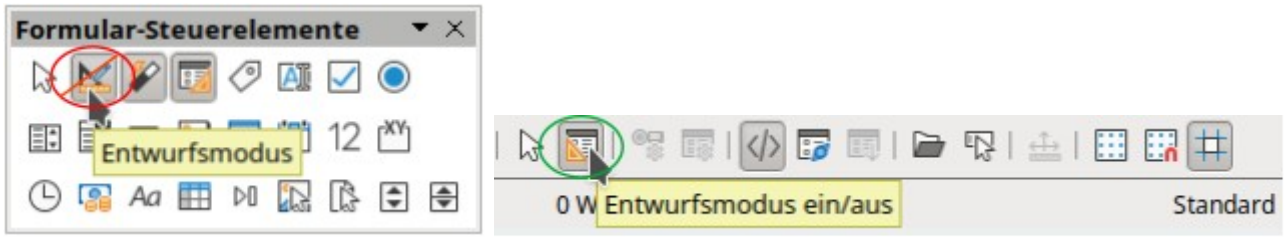
Einem einführenden Tag, der die Datei als XML-Datei kennzeichnet und zusätzlich noch den verwendeten Zeichensatz enthält, folgt ein Element, das die gesamten Daten umfasst. In der Regel gibt es zu jedem Element einen Starttag und einen Endtag, wobei der Endtag an dem / zu erkennen ist. Text, der zwischen 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.



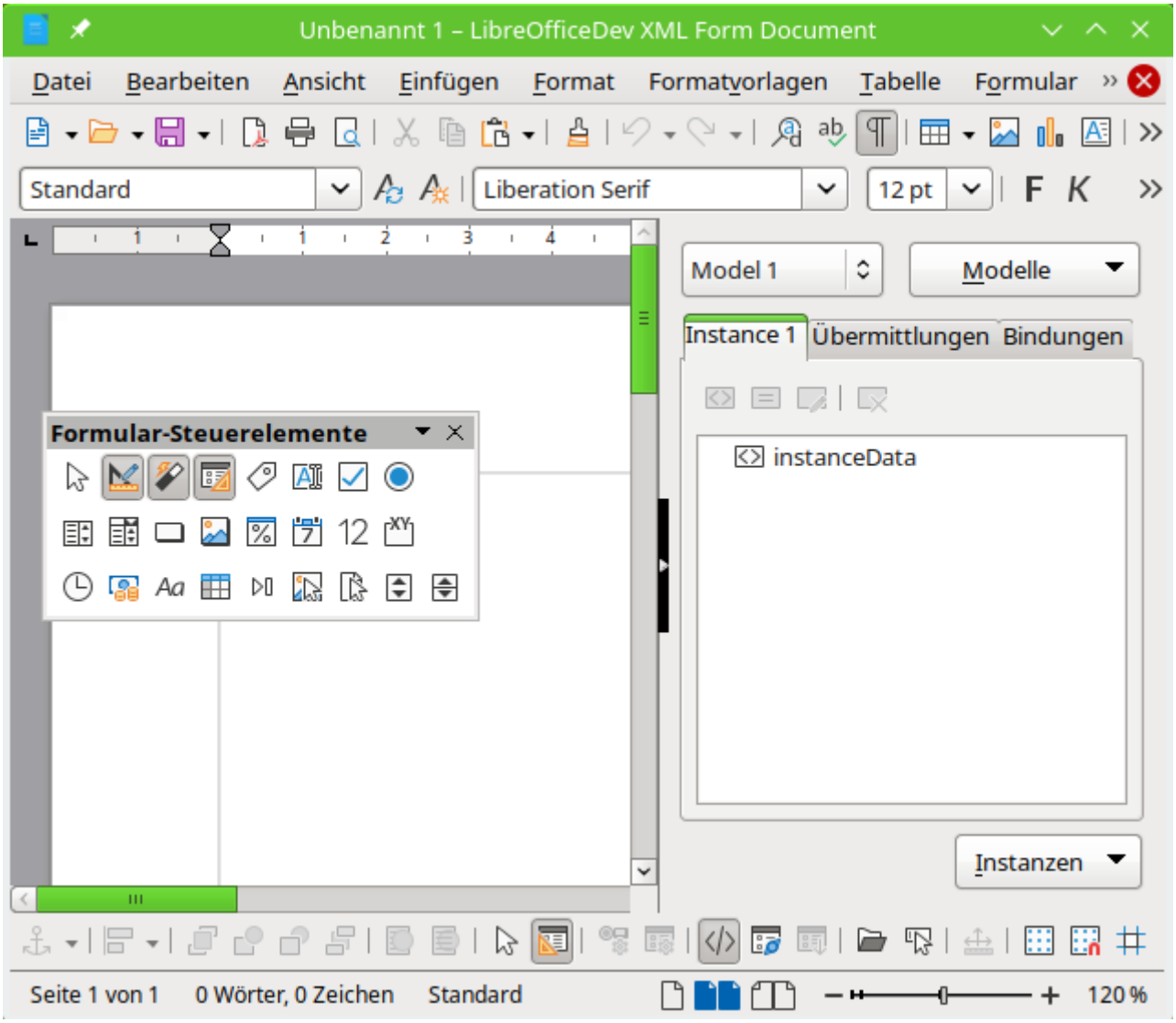
Das Formular wird im Entwurfsmodus geöffnet:



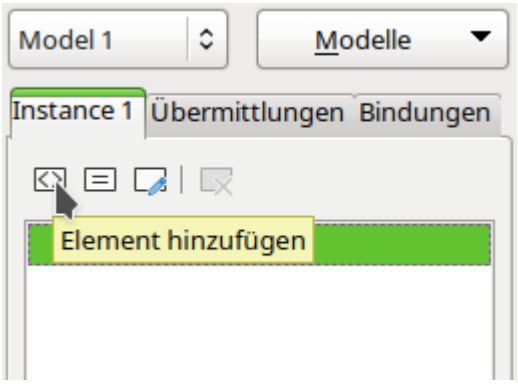
Wird das Dokument abgespeichert und erneut geladen, so wird das Formular zur Dateneingabe geöffnet. Zum weiteren Bearbeiten muss der **Entwurfsmodus ein/aus** eingeschaltet werden.

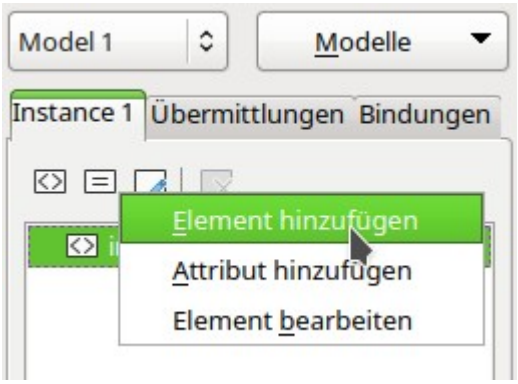
1 Siehe: *https://de.wikipedia.org/wiki/Extensible_Markup_Language*

Dieser Entwurfsmodus ist **nicht identisch** mit dem Entwurfsmodus, der über die Symbolleiste **Formular-Steuerelemente** oder über **Formular → Entwurfsmodus** erreicht werden kann.

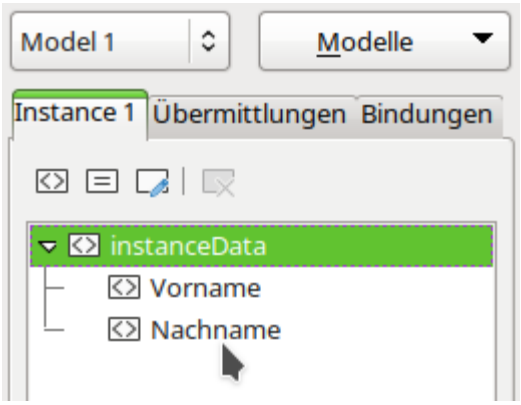


Wird die Symbolleiste für die Formular-Steuerelemente nicht angezeigt, so kann diese über **Ansicht → Symbolleisten → Formular-Steuerelemente** eingeblendet werden.

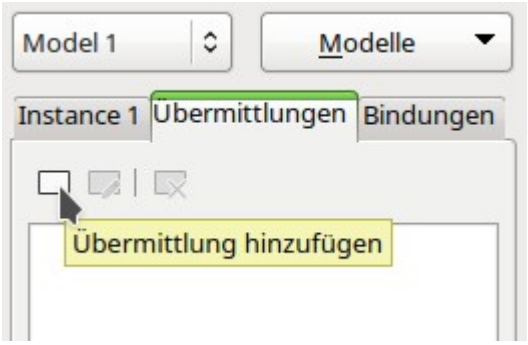




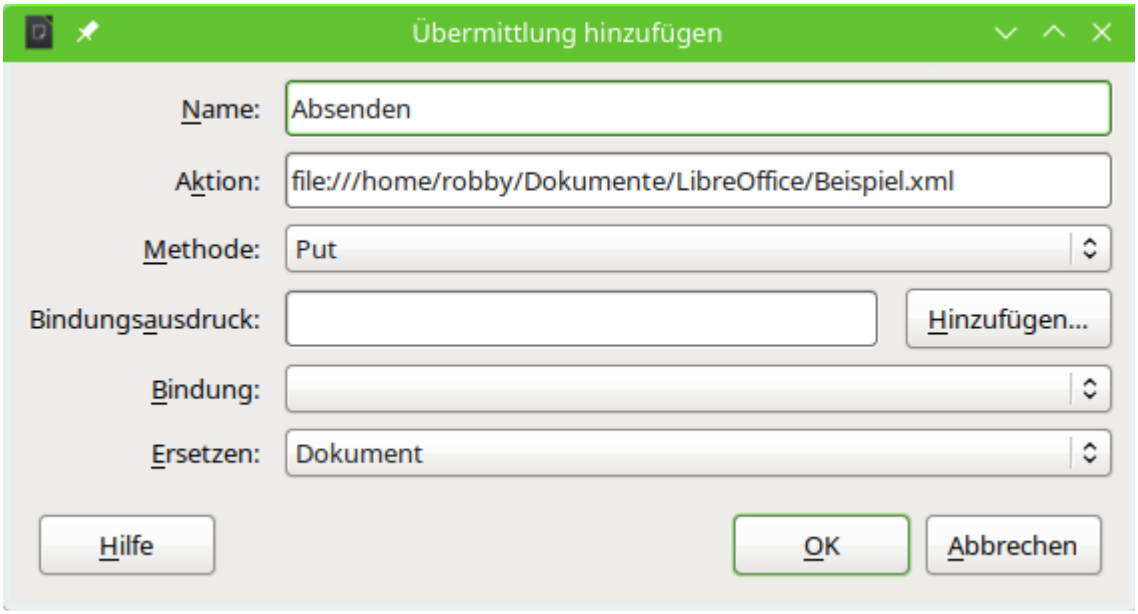
In dem Reiter **Instance 1** liegt bereits das umfassende Element **instanceData**. Diesem Element werden im folgenden zwei gleichberechtigte **Elemente** hinzugefügt.



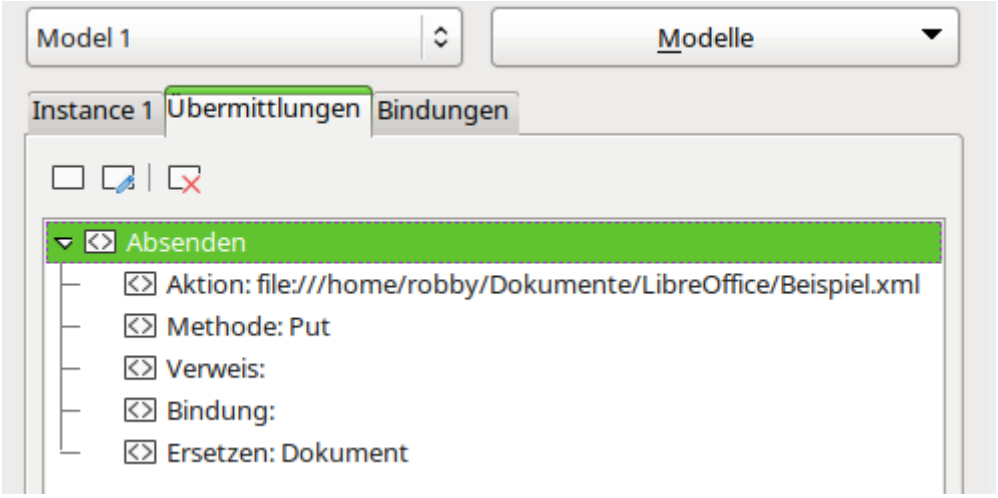
Der beim Hinzufügen der Elemente erscheinende Dialog wird auf jeden Fall in der Eigenschaft **Name** bearbeitet. Hier sind die Untergliederungen «Vorname» und «Nachname» hinzugefügt worden. Dadurch ist die komplette durch dieses Formular zu erstellende XML-Datei definiert. Die weiteren Eigenschaften des Dialogs können auch in den Bindungen oder im Formularfeld eingetragen werden. Sie werden durchgängig synchron gehalten.



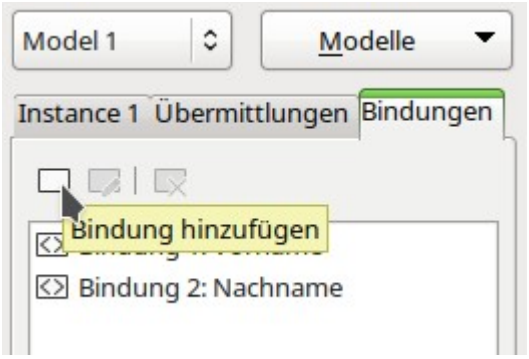
Im Reiter **Übermittlungen** wird eine Versandart hinzugefügt. In früheren Versionen von LibreOffice stand hier der englische Begriff «Submissions», für kurze Zeit auch die nicht passende deutschsprachige Übersetzung «Vorlagen».



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



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



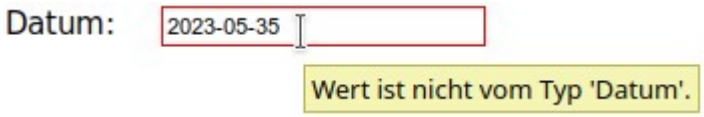
Bindungen werden standardmäßig direkt hinzugefügt. Nur so ist es möglich, die Formularfelder mit den entsprechenden Instanzen zu koppeln. So sind in obigem Screenshot bereits die beiden Bindungen für «Vorname» und «Nachname» zu sehen. Sollten die Bindungen nicht zu sehen sein so hilft es auch, das Formular zu speichern, zu schließen und neu zu starten.

Hilft dies auch nicht, so kann eine Bindung hinzugefügt, die den Angelpunkt zu den geplanten Formularfeldern darstellt.

Die Bindungen werden automatisch durchnummeriert, können natürlich auch mit separaten Namen versehen werden. So wurde oben die leere Bindung, die aus dem Klick auf «instanceData» entstanden ist, gelöscht und die Benennung der anderen Bindungen angepasst. Wird hier der gleiche Ausdruck wie bei den Instanzen als Bindungsausdruck eingegeben, so erfolgt die entsprechende Koppelung zu den Instanzen.

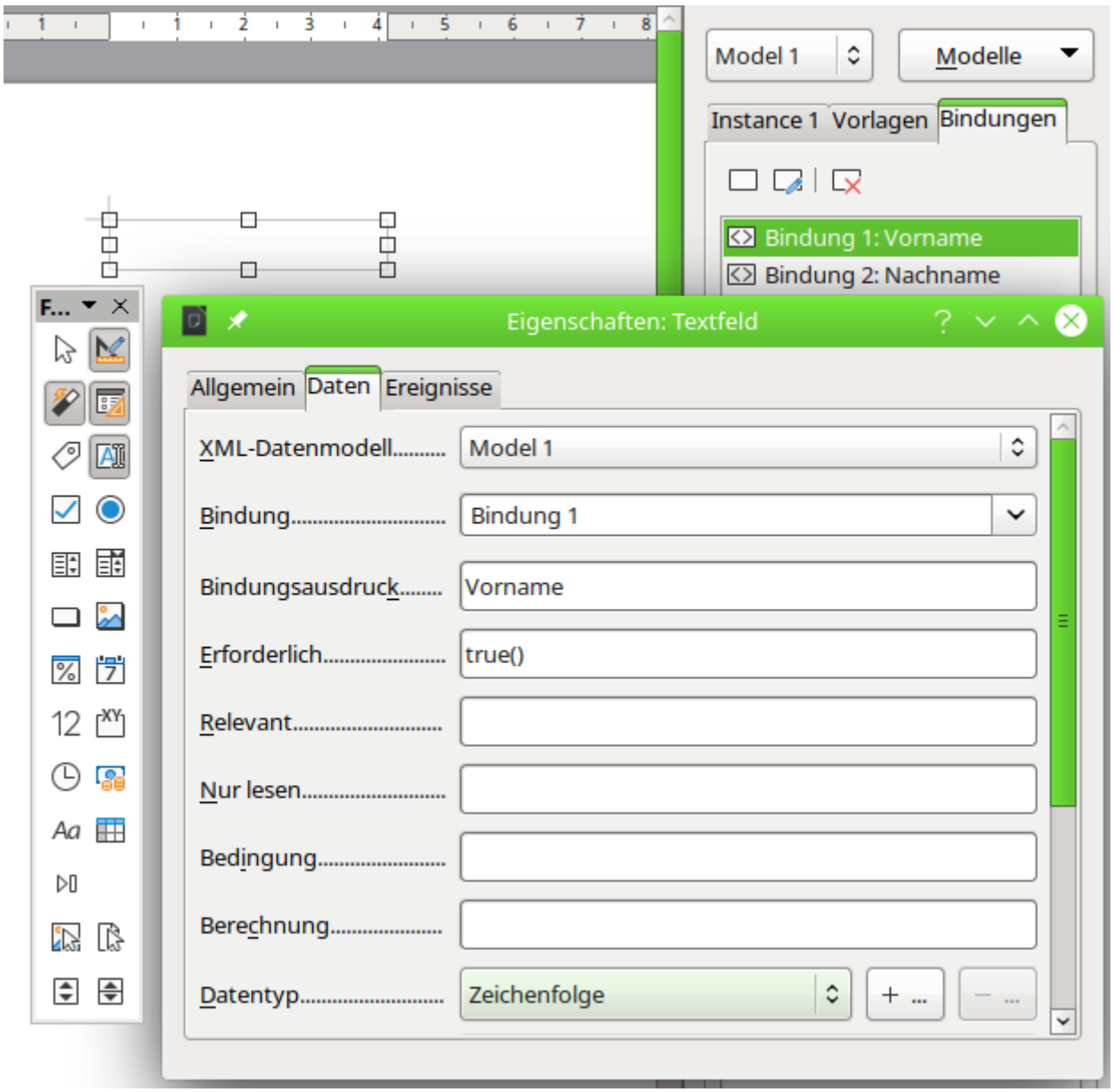
Jede Bindung wird mit entsprechenden Eigenschaften versehen. Diese können aber auch bereits bei der Festlegung der Instanzen mit dem gleichen Dialog erstellt werden. Gegebenenfalls können sie auch noch in den Formularfeldern nachgetragen werden.

Wichtig ist hier, dass **Einstellungen → Datentyp** mit einem entsprechenden Typ versehen wird. Erst so kann bei der Eingabe überprüft werden, ob die Eingabe einen Sinn ergibt. Bei einfachen **Zeichenketten** macht sich das nicht bemerkbar, aber falsch geschriebene Datumseingaben können so z. B. verhindert werden.



Wird später versucht, Inhalt mit einem anderen Datentyp über ein Formularfeld einzugeben, so wird das Formularfeld rot umrandet und beim Überfahren mit der Maus eine Meldung ausgegeben. Auch der Versand wird mit einer Fehlermeldung versehen. Allerdings schützt diese Einstellung letztlich nicht davor, bewusst falsch eingegebene Formate zu unterdrücken.

Für viele Funktionen besser geeignet ist daher das darauf zugeschnittene Formularfeld, das von vornherein die Dateneingabe steuert und falsche Werte gar nicht erst akzeptiert. Das Textfeld hingegen lässt erst einmal alle Eingaben zu und zeigt mit dem roten Rand, dass da etwas nicht stimmt.



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

Nur Formularfelder, die eine Verknüpfung zu dem Datenmodell ermöglichen, können auch ohne Makros genutzt werden. Dies sind:

Formularfeld	***Geeignet für …***
Textfeld	Das Allroundfeld für die Eingabe jeden Datentyps. Es kann alle Daten, die im XML-Formular eingelesen wurden, genau so wiedergeben, wie sie in der entsprechenden *.xml-Datei auch geschrieben werden. Nachteil ist, dass hier strikt die Formatvorgabe der *.xml-Datei beachtet werden muss. Ein Datum kann nicht in der deutschen Schreibweise eingegeben werden. Besonders kompliziert wäre die Eingabe eines Wertes für den Datentyp «Datum und Zeit».

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

Numerisches Feld	Nur für die Datentypen «Dezimal», «Doppelte Genauigkeit» («Double»), «Fließkomma», «Jahr», «Monat» und «Tag» geeignet. Es gibt grundsätzlich Zahlen mit 2 Nachkommastellen aus. Stören diese Angaben oder zeigen sie zu wenig Nachkommastellen, so sollte das formatierte Feld benutzt werden.
Zeitfeld	Nur für den Datentyp «Zeit» gedacht. Ein Zeitformat kann so vorgegeben werden dass entweder nur Stunden und Minuten oder Stunden, Minuten und Sekunden eingegeben werden können. Die Maximalzeit ist < 24:00:00.
Datumsfeld	Nur für den Datentyp «Datum» gedacht. Ein Datumsformat kann in vielen unterschiedlichen Schreibweisen vorgegeben werden. Außerdem kann das Feld zur Datumsauswahl mit der Einstellung **Aufklappbar → Ja** auf einen Kalender zugreifen.
Formatiertes Feld	Das Allroundfeld schlechthin. Hier sind alle Datentypen möglich. Auch die entsprechenden Formatierung kann erstellt werden. Es ist das einzige Feld, das dem Datentyp «Datum und Zeit» eine Eingabe ermöglicht, die nicht an die *.xml-Vorlage gehalten ist. Es ist das einzige Feld, bei dem für Zahleneingaben die Anzeige der Nachkommastellen beeinflussen kann. Es ist auch das einzige Feld, das Währungsangaben im Feld erlaubt und daraus einfache Dezimalzahlen für den xml-Export erstellt.
Markierfeld	Nur für den Datentyp «Ja/Nein (Boolean)» geeignet. Als Referenzwert sollte schlicht '1' für 'Ja' und '0' für 'Nein' vorgewählt werden.
Optionsfeld	Nur für den Datentyp «Ja/Nein (Boolean)» geeignet. Bei gleicher Bezeichnung werden die Optionsfelder zusammen gruppiert und nur eins der Felder gibt einen entsprechenden Wert an das XML-Formular weiter.
Listenfeld	Das Listenfeld kann direkt über **Allgemein → Listeneinträge** mit Einträgen versorgt werden. Es kann auch in XML-Formularen über ein zusätzliches Datenmodell, das die auswählbaren Einträge enthält, bestückt werden. Siehe dazu das Kapitel *Formularfelder*. Eine Auswahl mehrere Werte über **Allgemein → Mehrfachauswahl** ist nicht möglich.
Kombinationsfeld	Sind neben auswählbaren Einträgen auch neue Einträge gewünscht, so ist das Kombinationsfeld passend. Das Kombinationsfeld kann direkt über **Allgemein → Listeneinträge** mit Einträgen versorgt werden. Es kann auch in XML-Formularen über ein zusätzliches Datenmodell, das die auswählbaren Einträge enthält, bestückt werden. Siehe dazu das Kapitel *Formularfelder*.
Schaltfläche grafische Schaltfläche	Die Schaltfläche ist zum Versand der Daten notwendig. Beide Schaltflächen bieten eine Verbindung zu den vorgesehenen Versandarten an.

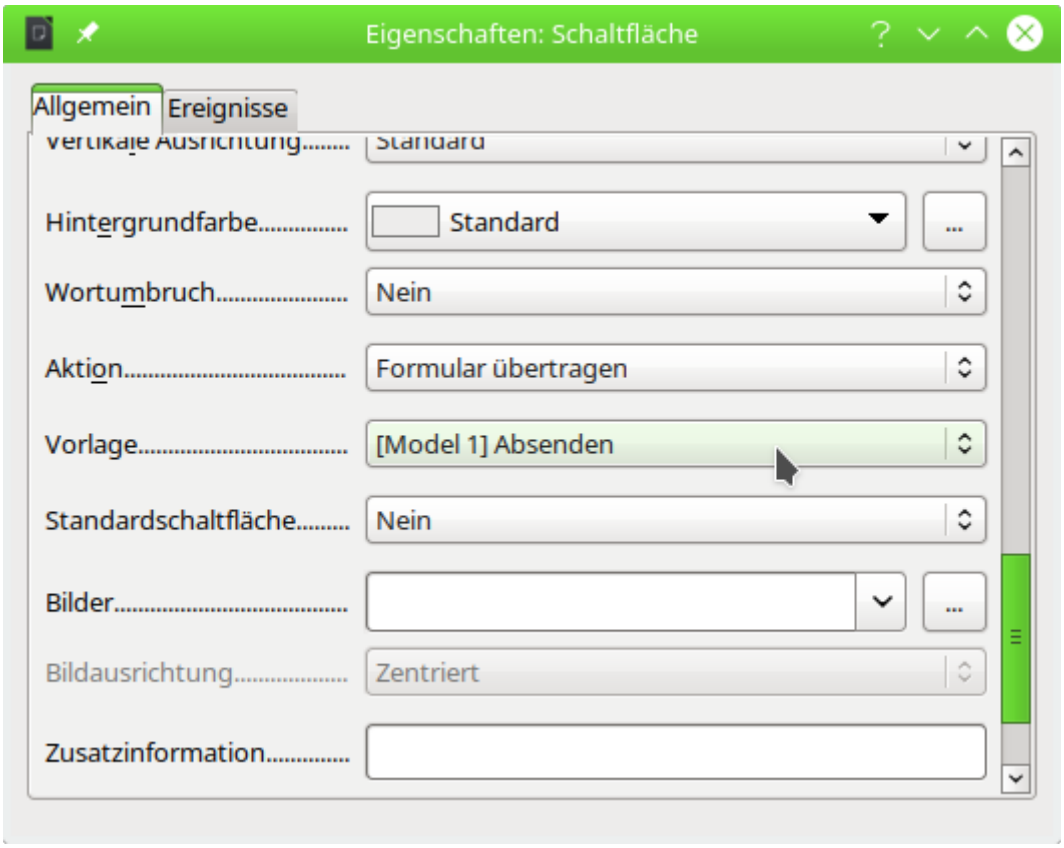
Hinweis

Die Formular-Steuerelemente werden hier nicht kleinschrittig erklärt. Hierzu sollte das Base-Handbuch zu Rate gezogen werden.
Assistenten, die bei Base-Formularen funktionieren, starten bei XML-Formularen nicht. Sie wurden speziell auf die Zusammenarbeit mit Datenbanken erstellt.

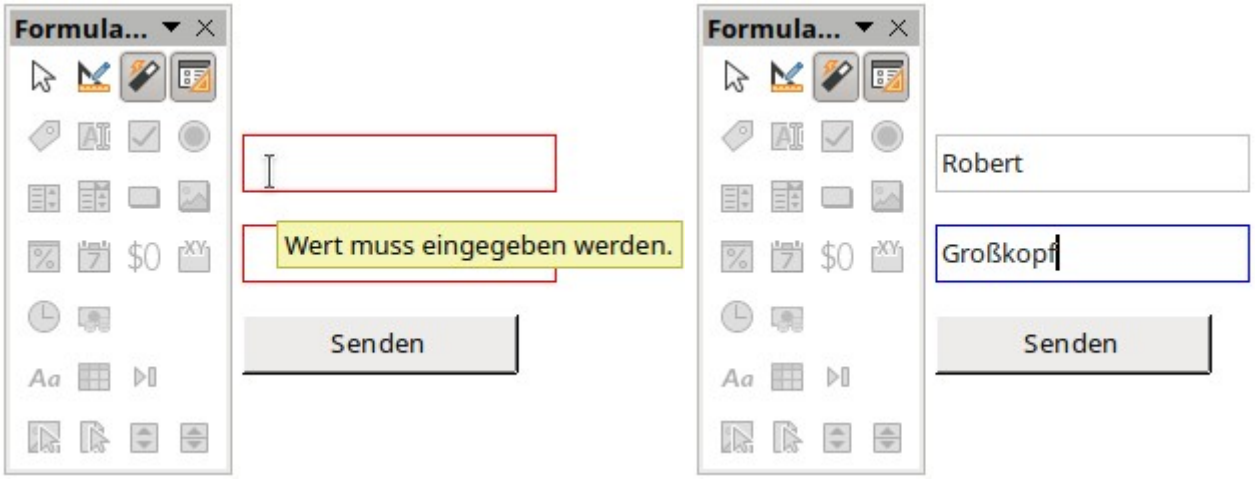
Tipp

Lässt sich ein Formular-Steuerelement nicht aufziehen, so kann dies an der fehlenden Formulardefinition liegen. Über den Formularnavigator muss, wie bei Base-Formularen, ein Formular erstellt werden – auch wenn es keinen Inhalt hat. Dieses Formular sollte bei XML-Formulardokumenten allerdings automatisch erstellt werden.

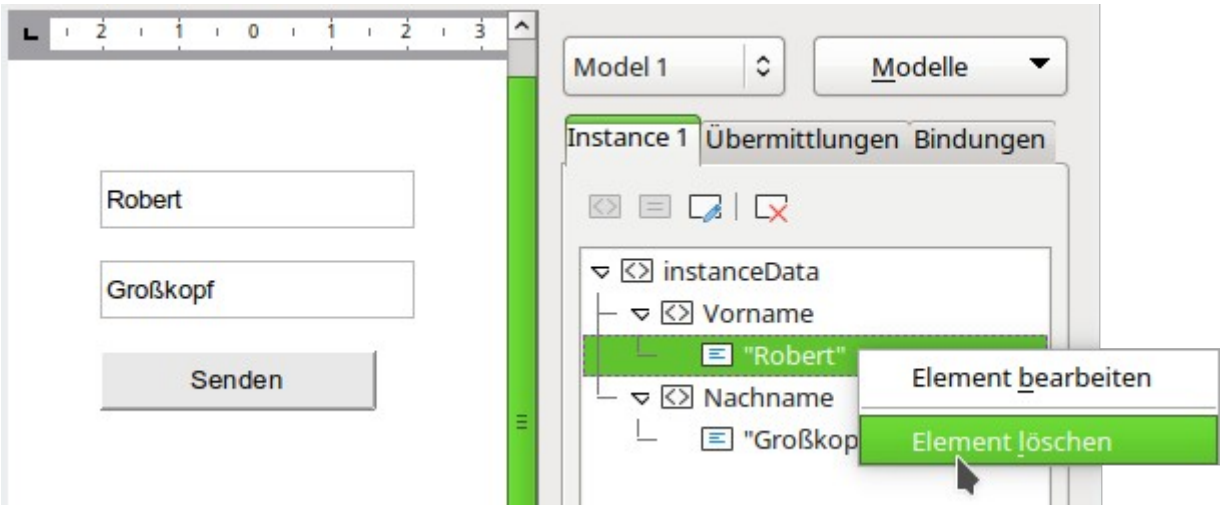
In diesem Beispiel steht nur das **Model 1** zur Verfügung. **Bindung** lässt mit einem Auswahlfeld die Auswahl der beiden definierten Bindungen zu, hier also von **Bindung 1**. Die restlichen Einträge spiegeln die Definition der Bindung wieder.



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



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. Ist eine Eingabe erfolgt und steht der Cursor nicht in dem Feld, so wird die Umrandung in der voreingestellten Farbe gezeigt. Steht der Cursor noch in dem Feld, so erscheint das Feld mit blauer Umrandung.



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 über ein Makro für alle Felder oder jeweils über das Löschen des entsprechenden Elementes in der Instanz erfolgen. Der einfache Nutzer bekommt allerdings diesen Datennavigator nicht zu Gesicht und kann die Elemente in dem Formular entsprechend nur leeren.

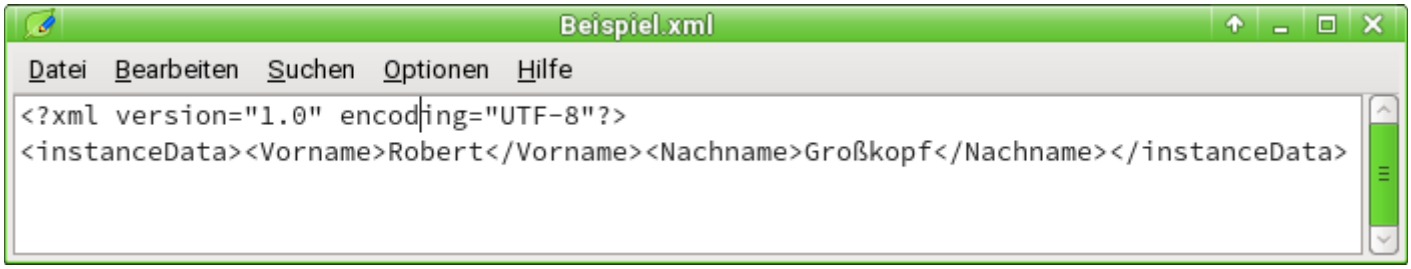
Hinweis

Änderungen am Formularinhalt werden nicht automatisch vom Writer registriert. Sie werden trotzdem über **Datei → Speichern** mit abgespeichert. Ohne diese Speicherung erscheint das Formular beim nächsten Mal wieder mit den alten Daten.

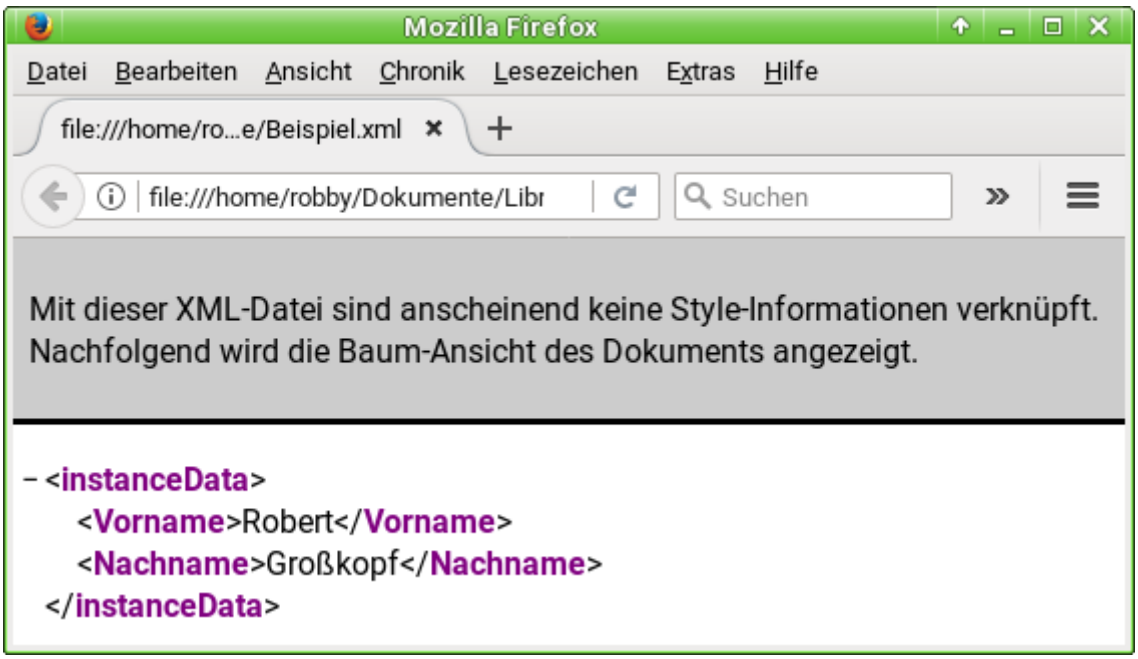
```
<xforms:model id="Model 1">
    <xforms:instance id="Instance 1">
        <instanceData>
            <Vorname>Robert</Vorname>
            <Nachname>Großkopf</Nachname>
        </instanceData>
    </xforms:instance>
    <xforms:bind id="Bindung 1" nodeset="Vorname" required="true()" type="xsd:string"/>
    <xforms:bind id="Bindung 2" nodeset="Nachname" required="true()" type="xsd:string"/>
    <xforms:submission id="Absenden"
            action="file:///home/robby/Dokumente/LibreOffice/xform/Daten_Start.xml"
            method="put" indent="false" omit-xml-declaration="false" standalone="false"
            replace="none"/>
    <xsd:schema/>
</xforms:model>
```

Die Formulardatei ist eine gepackte Datei. In ihr befinden sich viele verschiedene Dateien, unter anderem auch die Datei «content.xml». In dieser «content.xml» ist das gesamte Formular verzeichnet. Auch die Daten sind dort im xml-Format abgespeichert. Wird also nur das Formular benötigt und ist keine weitere Datenverarbeitung geplant, so reicht es aus, das Formular ohne

jeden Eintrag für das Absenden einfach per **Datei → Senden → Dokument als E-Mail…** weiter zu schicken.



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:

Damit Daten formatiert angezeigt werden muss die *.xml-.Datei auf eine *.xls-Datei zugreifen. Außerdem muss die *.xml-Datei über einen Webserver aufgerufen werden. Beim direkten Aufruf im Dateisystem funktioniert die Verbindung nicht.

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:

```
001 <?xml version="1.0" encoding="UTF-8"?>
002 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
003    <xsl:template match="/">
004       <HTML>
005          <Head>
006             <Title>XML-Formular</Title>
007          </Head>
008          <Body>
009             <H2>Daten aus dem XML-Formular</H2>
010             <table>
011                <tr>
012                   <th>Nachname</th>
013                   <th>Vorname</th>
014                </tr>
015                <xsl:for-each select="Data/instanceData">
016                <xsl:sort select="Nachname" order="ascending" data-type="text" />
017                <tr>
018                   <td><xsl:value-of select="Nachname"/></td>
019                   <td><xsl:value-of select="Vorname"/></td>
020                </tr>
021                </xsl:for-each>
022             </table>
023          </Body>
024       </HTML>
025    </xsl:template>
026 </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:

```
001 <?xml version="1.0" encoding="UTF-8"?>
002 <?xml-stylesheet type="text/xsl" href="./Daten_Start_formatiert.xsl" ?>
003 <Data>
004 <instanceData><Vorname>Robert</Vorname><Nachname>Großkopf</Nachname></instanceData>
005 <instanceData><Vorname>Jutta</Vorname><Nachname>Schön</Nachname></instanceData>
006 <instanceData><Vorname>Ammon</Vorname><Nachname>Düül</Nachname></instanceData>
007 <instanceData><Vorname>Nick</Vorname><Nachname>Knatterton</Nachname></instanceData>
008 <instanceData><Vorname>Silvia</Vorname><Nachname>Bach</Nachname></instanceData>
009 <instanceData><Vorname>Rob</Vorname><Nachname>van Buur</Nachname></instanceData>
010 <instanceData><Vorname>Bob</Vorname><Nachname>Baumeister</Nachname></instanceData>
011 </Data>
```

In der zweiten Zeile dieser Datei wird darauf hingewiesen, dass die Darstellung in einer separaten Datei «Daten_Start_formatiert.xsl» definiert ist. Diese Datei liegt direkt im gleichen Ver-

zeichnis wie die *.xml-Datei. 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 **Vorlagen (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]

```
001 SUB DatenNachXmlDatei(oEvent AS OBJECT)
002    DIM a()
003    DIM b()
004    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».

```
005    oDoc = ThisComponent
006    stDir = Left(oDoc.Location,Len(oDoc.Location)-Len(oDoc.Title))
007    stFileTmp = stDir & "Daten_Start.xml"
008    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.

```
009    oFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
010    IF oFileAccess.exists(stFileTmp) THEN
011       oInputStream = createUnoService("com.sun.star.io.TextInputStream")
012       oFile = oFileAccess.OpenFileReadWrite(stFileTmp)
013       oInputStream.SetInputStream(oFile.getInputStream)
014       DO WHILE NOT oInputStream.isEOF
015          stTmp = stTmp & oInputStream.ReadLine()
016       LOOP
017       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.

```
018       a = Split(stTmp,"?>")
019       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.

```
020       IF oFileAccess.exists(stFileData) THEN
021          oInputStream = createUnoService("com.sun.star.io.TextInputStream")
022          oFileData = oFileAccess.OpenFileReadWrite(stFileData)
023          oInputStream.SetInputStream(oFileData.getInputStream)
024          DO WHILE NOT oInputStream.isEOF
```

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

```
025             stDataOld = stDataOld & oInputStream.ReadLine() & CHR(13) & CHR(10)
026         LOOP
027         oInputStream.closeInput
```

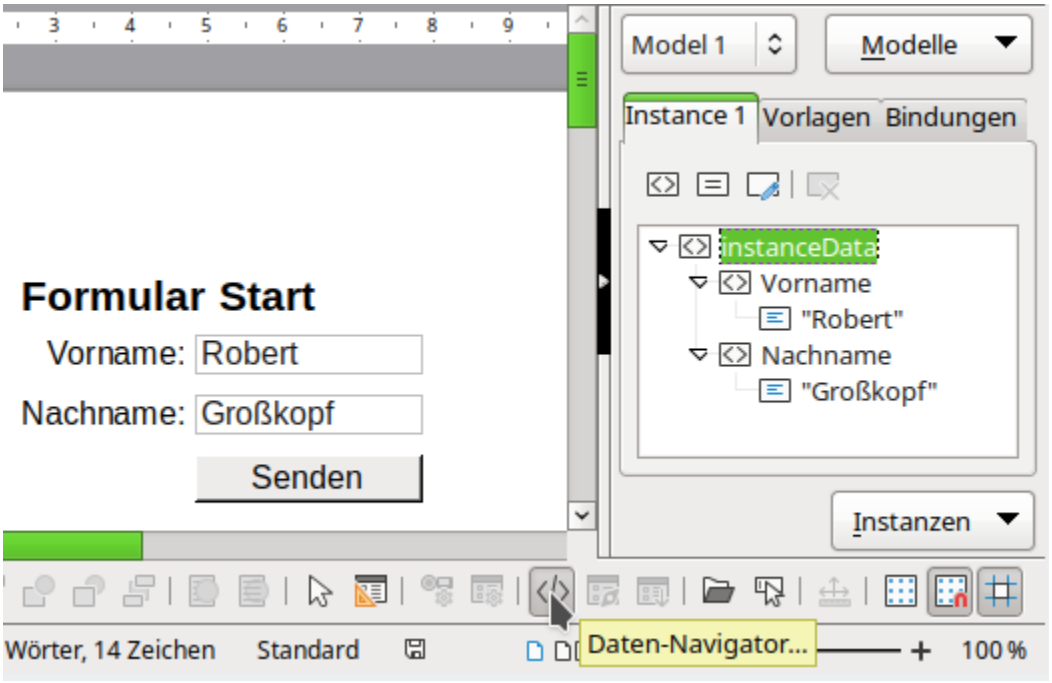
Die in die Datei zu schreibenden Daten werden aus den bestehenden Daten und der neuen Zeile neu zusammengestellt. Die Datei wird anschließend neu geschrieben. Zum Schluss wird die Ausgabe in die Datei wieder geschlossen.

```
028         oOutputStream = createUnoService("com.sun.star.io.TextOutputStream")
029         b = Split(stDataOld,"</" & stInstance & ">")
030         stDataNew = b(0) & stTmpData & CHR(13) & CHR(10) & "</" & stInstance & ">"
031         oFileData = oFileAccess.OpenFileReadWrite(stFileData)
032         oOutputStream.SetOutputStream(oFileData.getOutputStream)
033         oOutputStream.writeString(stDataNew)
034         oOutputStream.closeOutput()
035       END IF
036    END IF
037 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 kann dann nicht mehr über die Funktion zum Einblenden der Seitenleiste angezeigt werden. 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 **Vorlagen(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

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.

Das **Ersetzen** erfolgt hier, wie bei allen anderen Methoden, nachdem der ursprüngliche Versand erfolgt ist.

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:

```
001 <?php
002 //Datei xmldata.php
003 $datafile = file("php://input");
004 ?>
```

Der Inhalt sieht dann so aus:

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

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

```
001 <?php
002 //Datei xmldata.php
003 $datafile = file("php://input");
004 $dataall = "data/Daten.xml";
005 IF (count($datafile) > 1) {
006    $data = $datafile[1];
007    $rootNode = substr($data, 1, strpos($data, ">")-1);
008    $data = substr($data, strpos($data, ">")+1, strpos($data,
          "</".$rootNode.">")-strlen("<".$rootNode.">"));
009    IF (file_exists($dataall)) {
010       $datei_arr = file($dataall) ;
011       array_pop($datei_arr);
012       $datei_arr[] = $data."\r\n</".$rootNode.">";
013       $datei = fopen($dataall, "w+");
014       foreach($datei_arr as $zeile){
015          fwrite($datei, $zeile);
```

4 Siehe hierzu das Beispiel «Formular_einfach.odt»

```
016         }
017         fclose($datei);
018      }
019      ELSE {
020         $datei = fopen($dataall, "a");
021         fwrite($datei, $datafile[0]."<".$rootNode.">\r\n".$data."\r\n</".
               $rootNode.">");
022      }
023      fclose($datei);
024   }
025   ?>
```

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 «Daten.xml» liegt in dem obigen Beispielcode in einem Unterverzeichnis «data» (Zeile 4).

Die eigentlichen Daten liegen in der ankommenden Datei zwischen den Nodes für die Hauptinstanz (Wurzelelement, Zeile 8).

Existiert diese Datei nicht, so wird sie ab Zeile 19 mit dem Inhalt der abgesandten Datei, um einige Zeilenumbrüche erweitert, abgespeichert. Die Zeilenumbrüche sind notwendig, damit bei folgenden zusätzlichen Datensätzen der Abschluss der gemeinsamen Hauptinstanz (z. B. **</instanceData>**) jeweils einfacher ersetzt werden kann.

Existiert die Datei wird ab Zeile 9 in ein PHP-Array zeilenweise ausgelesen. Das letzte Element des Arrays, das den Abschlusstag enthält, wird aus dem Array gelöscht (Zeile 11). Danach wird der neue Datensatz sowie der Abschlusstag, mit einem Zeilenumbruch, an das Array angehängt (Zeile 12). Zum Schluss wird die Datei wieder komplett neu aufgebaut. Das Array wird dabei Zeile für Zeile in die Datei übertragen.

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_get.php?Vorname=Lieschen&Nachname=M%FCller

Mit der GET-Methode werden also lediglich die Daten direkt weiter gegeben. Allerdings passiert es, dass bei der GET-Methode die entsprechende Seite auf dem Webserver mehrmals mit Daten versorgt wird. Hier muss mit Hilfe des aufnehmenden Scripts und den entsprechenden Servervariablen gegengesteuert werden.

```
001   <?php
002   //Datei xmldata_get.php
003   $dataall = "data/Daten_einfach_formatiert.xml";
004   IF ($_SERVER['REQUEST_METHOD'] == "HEAD"){
005      $vorname = $_GET["Vorname"];
006      $nachname = $_GET["Nachname"];
007      $data = "<Name><Vorname>".$vorname."</Vorname><Nachname>".
            $nachname."</Nachname></Name>";
008      IF (file_exists($dataall)) {
009         $datei_arr = file($dataall) ;
010         $rootNode = $datei_arr[count($datei_arr)-1];
011         array_pop($datei_arr);
012         $datei_arr[] = $data."\r\n".$rootNode;
013         $datei = fopen($dataall, "w+");
014         foreach($datei_arr as $zeile){
015            fwrite($datei, $zeile);
016         }
017         fclose($datei);
018      }
019   }
020   ?>
```

Die Daten werden hier in einer bereits auf dem Server liegenden Datei «Daten_einfach_formatiert.xml» abgespeichert. Das hat den Vorteil, dass dort auch direkt eine *.xsl-Datei eingebunden werden kann. Der Datei müssen also lediglich die Daten jeweils hinzugefügt werden.

Über die **REQUEST_METHOD → HEAD** wird dafür gesorgt, dass das Script nur einmal ausgeführt wird. Hier wäre ab LO 7.6 auch die **REQUEST_METHOD → GET** möglich, die in vorhergehenden Versionen leider für einen doppelten Versand verantwortlich war. Ab Zeile `5` werden die Variablen ausgelesen, die von dem Formular über die URL weiter gegeben wurden. Die Variablenbezeichnungen entsprechen den Elementsbezeichnungen in der Instanz. Die ermittelten Werte werden anschließend zu einer Zeile als **$data** zusammengefasst.

Einträge in die vorliegende Datei werden nur ausgeführt, wenn die Datei bereits existiert (Zeile `8` ff.). In Zeile `10` wird die Abschlussbezeichnung für die Hauptinstanz (Wurzelelement) ausgelesen, bevor sie anschließend aus dem Code entfernt wird. So kann die neue Zeile eingefügt und der Abschluss der Hauptinstanz nach einem Zeilenumbruch wieder angehängt werden. Wie beim POST-Befehl wird die Datei komplett neu Zeile für Zeile geschrieben.

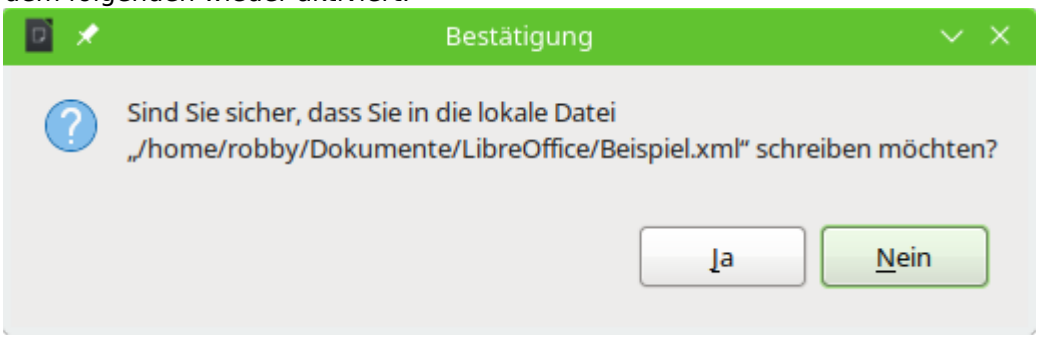
Auch hier sollte, wie bei der POST-Methode, der Eintrag **Submissions → Ersetzen → Kein** gesetzt werden.

PUT-Methode

Hier wird eine lokale Adresse eingetragen. Es wird eine Datei lokal geschrieben und abgelegt.

Vorsicht

Wird die Datei nicht rechtzeitig weiter verarbeitet, so wird der Inhalt mit einem erneuten Abschicken des Formulars überschrieben. Deswegen war diese Funktion zwischenzeitig völlig außer Kraft gesetzt und wird erst seit LO 7.6 mit einem Warn-Dialog wie dem folgenden wieder aktiviert:



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[5]. 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:

```
001  SUB Mailversand
002     DIM attachs(0)
003     IF GetGuiType() = 1 THEN
004        oMailer = createUnoService("com.sun.star.system.SimpleSystemMail")
005        ' Sonst Linux/Mac
006     ELSE
```

5 Als Hilfe zur Ermittlung der richtigen URL-Schreibweise dient die den Beispielen beigefügte Datei «URL_ermitteln.odt»

```
007         oMailer = createUnoService("com.sun.star.system.SimpleCommandMail")
008     END IF
009     oMailProgram = oMailer.querySimpleMailClient()
010     oMessage = oMailProgram.createSimpleMailMessage()
011     oMessage.setRecipient("lieschen.mueller@example.org")
012     oMessage.setSubject("XML-Daten")
013     attachs(0) = "file:///home/user/Dokumente/datendatei.xml"
014     oMessage.setAttachement(attachs())
015     oMailProgram.sendSimpleMailMessage(oMessage, 0 )
016 END SUB
```

Versand des gesamten Formulars per Mail

Natürlich kann nicht nur die Datendatei an eine Mail angehängt werden. Es ist auch möglich, das ganze Formular per Mail weiter zu versenden.[6] Ein Formular für so einen Mailversand könnte so aussehen:

Die einfachste Methode hierzu ist im Writer der Weg über **Datei → Senden → Dokument als E-Mail…** . Komfortabler ist aber, solch ein Formular erst einmal auf die Vollständigkeit hin zu überprüfen, dann die Eingaben mit einem Schreibschutz zu versehen und anschließend zu versenden. Dadurch können nicht im Nachhinein versehentlich Daten zu dem Fortbildungsantrag verändert werden.

```
001 SUB Submit_Mail
002     DIM oDoc AS OBJECT
003     DIM oController AS OBJECT
004     DIM oDrawpage AS OBJECT
005     DIM oField AS OBJECT
006     DIM oPath AS OBJECT
007     DIM oMailer AS OBJECT
008     DIM oMailProgram AS OBJECT
009     DIM oMessage AS OBJECT
010     DIM oUserData AS OBJECT
011     DIM oUser AS OBJECT
012     DIM oFileAccess AS OBJECT
013     DIM stTitle AS STRING
014     DIM stTimestamp AS STRING
015     DIM stPrintDir AS STRING
016     DIM stName AS STRING
017     DIM stMail AS STRING
018     DIM stUser AS STRING
019     DIM ar()
020     oDoc = ThisComponent
021     oController = oDoc.getCurrentController()
022     oDrawpage = oDoc.Drawpage
023     stTitle = oDoc.DocumentProperties.Title
024     REM Erforderliche Felder ueberprüfen
025     FOR i = 0 TO oDrawpage.Count - 1
```

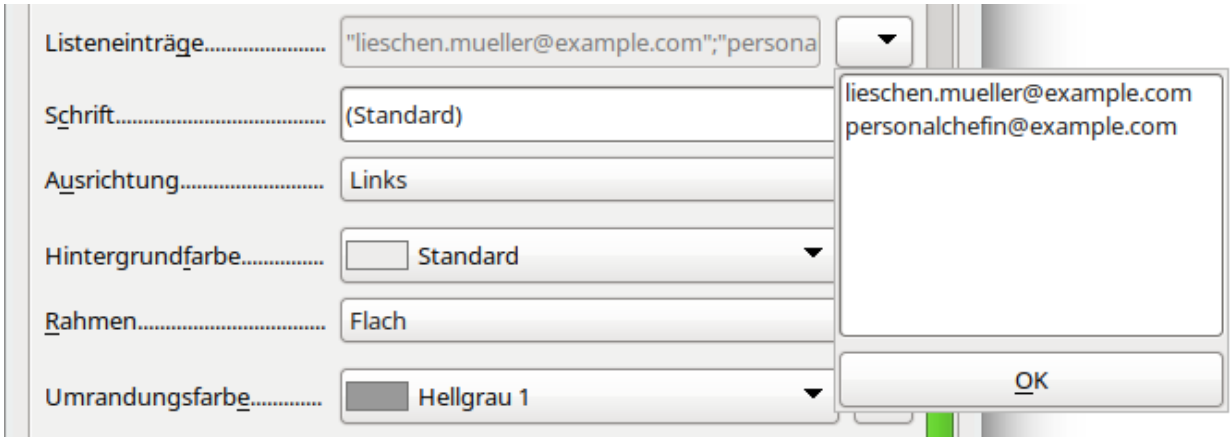
6 Siehe hierzu das Beispiel «Fortbildungsantrag.odt»

```
026         oField = oDrawpage.getByIndex(i).Control
027         IF oField.supportsService("com.sun.star.form.DataAwareControlModel") THEN
028            IF oField.Validator.RequiredExpression = "true()" AND
                  oField.Validator.getValue("string") = "" THEN
029               msgbox ("Die Einträge sind noch nicht vollständig." & CHR(13) &
                     "Das Formular kann noch nicht versandt werden", 16, "Mailversand")
030               EXIT SUB
031            END IF
032         END IF
033      NEXT
034      REM oDoc mit Timestamp abspeichern, sofern noch kein Timestamp enthalten
035      stTimestamp = Year(Now()) & "-" & Right("0" & Month(Now()),2) & "-" &
            Right("0" & Day(Now()),2) & "_" & Right("0" & Hour(Now()),2) & "-" &
            Right("0" & Minute(Now()),2)
036      oPath = createUnoService("com.sun.star.util.PathSettings")
037      stTempPath = oPath.Temp & "/LoFb"
038      REM vorher erstellte temporäre Dateien löschen, indem das Verzeichnis dafür
            gelöscht wird
039      oFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
040      IF oFileAccess.exists(stTempPath) THEN
041         oFileAccess.kill(stTempPath)
042      END IF
043      REM Speicherung direkt im gleichen Verzeichnis:
044      REM stDir = Left(oDoc.Location,Len(oDoc.Location)-Len(oDoc.Title))
             ' Der Titel des Formulars wird von der URL abgetrennt.
045      IF Len(oDoc.Title) > 22 THEN
046         stPrintDir = stTempPath & oDoc.Title
047      ELSE
048         stPrintDir = stTempPath & "/" & stTitle & "_" & stTimestamp & ".odt"
049      END IF
050      oDoc.StoreAsURL(stPrintDir,ar())
051      REM Ausgefüllte Felder mit Schreibschutz versehen, Namen auslesen
052      FOR i = 0 TO oDrawpage.Count - 1
053         oField = oDrawpage.getByIndex(i).Control
054         IF oField.supportsService("com.sun.star.form.DataAwareControlModel") THEN
055            IF oField.Validator.getValue("string") <> "" AND
                  oField.Validator.getValue("string") <> "0" THEN
056               oController.getControl(oField).Model.Enabled = False
057            END IF
058            IF oField.Name = "txt_Name" THEN
059               stName = oField.Text
060            END IF
061            IF oField.Name = "lbo_Mail" THEN
062               stMail = oField.CurrentValue
063            END IF
064         END IF
065      NEXT
066      oDoc.Store()
067      REM Benutzerdaten für den Absender auslesen
068      DIM arUser(0) AS NEW com.sun.star.beans.PropertyValue
069      oUserData = createUnoService("com.sun.star.configuration.ConfigurationProvider")
070      arUser(0).Name = "nodepath"
071      arUser(0).Value = "org.openoffice.UserProfile/Data"
072      oUser = oUserData.createInstanceWithArguments
            ("com.sun.star.configuration.ConfigurationAccess", arUser())
073      stUser = oUser.givenname & " " & oUser.sn
074      REM Mail zusammenstellen und versenden
075      DIM attachs(0)
076      IF GetGuiType() = 1 THEN
077         oMailer = createUnoService("com.sun.star.system.SimpleSystemMail")
078      ' Sonst Linux/Mac
079      ELSE
080         oMailer = createUnoService("com.sun.star.system.SimpleCommandMail")
081      END IF
082      oMailProgram = oMailer.querySimpleMailClient()
083      oMessage = oMailProgram.createSimpleMailMessage()
```

```
084     oMessage.setRecipient(stMail)
085     oMessage.setSubject("Fortbildungsantrag von " & stName)
086     oMessage.Body = "In der Anlage finden Sie den ausgefüllten Fortbildungsantrag."
           & CHR(13) & CHR(13) & "Mit freundlichen Grüßen" & CHR(13) & CHR(13) & stUser
087     attachs(0) = stPrintDir
088     oMessage.setAttachement(attachs())
089     oMailProgram.sendSimpleMailMessage(oMessage,0)
090 END SUB
```

Die Weiterleitung als Mailanhang erfolgt in 5 Schritten.

1. Die Felder werden darauf hin untersucht, ob auch alle notwendigen Felder ausgefüllt worden sind. Hier werden die Eigenschaften des XML-Formulardokumentes **`requiredExpression = "true()"`** abgefragt und geklärt, ob eventuell **`getValue("string")`** bei einem erforderlichen Feld leer ist. Über diese Funktion können alle Felder erreicht werden, während die Nachfrage nach **`oField.CurrentValue`** von den benutzten Kontrollfeldern abhängig ist. (Zeilen `23` bis `31`)
   Ist ein erforderlicher Eintrag noch nicht vorhanden, so wird die Prozedur über **`EXIT SUB`** nach Ausgabe einer Fehlermeldung direkt beendet. (Zeilen `27` und `28`)
2. Das Formular wird im temporären Verzeichnis abgespeichert. Dabei wird dem vorgesehenen Namen, ermittelt aus **Datei → Eigenschaften → Beschreibung → Titel**, ein Zeitstempel hinzugefügt. (Zeilen `33` bis `48`)
   Wird das Formular mit dem entsprechenden Zeitstempel von der nächsten Person aufgerufen, so wird über die Länge des Dateinamens bestimmt, ob bereits ein Zeitstempel enthalten ist. Dann wird einfach der aktuelle Dateiname beim Abspeichern übernommen (Zeile `47`).
   Alte Einträge im Temporären Verzeichnis werden mit Hilfe des zusätzlich erstellten Unterverzeichnisses gelöscht. (Zeilen `37` bis `40`)
3. Alle Felder, die bereits einen Inhalt haben, werden beim Export mit einem Schreibschutz versehen. Außerdem wird noch der Name der Person, für die der Antrag gestellt wird, ausgelesen, damit dieser anschließend auch in den Betreff der Mail eingefügt werden kann. (Zeilen `50` bis `65`)
   Diese Einstellungen werden nur in dem exportierten Formular geändert. Andernfalls würde das Originalformular immer wieder nach einer Speicherung verlangen und die Eingaben gesperrt werden. Deshalb muss erneut abgespeichert werden (Zeile `66`).
4. Die Benutzerdaten aus **Extras → Optionen → LibreOffice → Benutzerdaten** werden für die Erstellung des Mailinhaltes ausgelesen. (Zeilen `68` bis `73`)
5. Die Mail wird entsprechend zusammengestellt und an das Mailprogramm weiter geleitet. Die in Zeile `84` aufgeführte Mailadresse wurde aus einem Listenfeld in dem Formular ausgelesen, dessen Liste direkt im Formularfeld gespeichert wurde:



Komplexere Formulardefinition[7]

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.

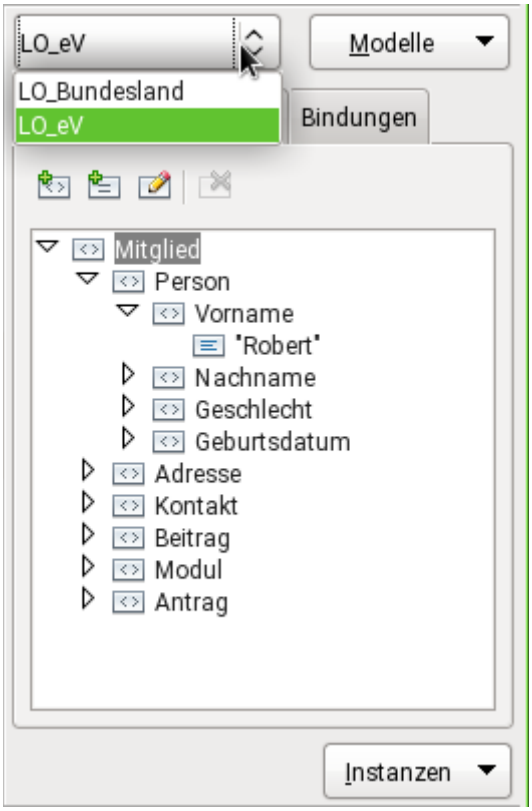
Der gesamte weitere Inhalt wird durch **Elemente** strukturiert. Dabei werden für **wohlgeformte** XML-Dateien folgende Punkte definiert[8]:

- Die XML hat genau ein **Wurzelelement**. Das äußere Element heißt in der Standardeinstellung 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.
- 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. Zwar ist es möglich, mehrere Instanzen zu erstellen, nur lässt sich leider keine richtige Bindung zu den Feldern der Instanzen erstellen, so dass auch keine Dateneingabe in Felder dieser Instanzen bisher möglich ist. 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, Übermittlungen (Submissions) und Bindungen neu. In einem Formular können mehrere Modelle existieren. Es kann von Modell zu Modell zum Bearbeiten umgeschaltet werden.

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



Das obige Modell «LO_eV» hat eine Instanz «Mitglied». Standardmäßig ist das Root-Element «instanceData». Die Umbenennung des Root-Elements ist erst mit den LO-Versionen 7.4 und neuer möglich.

Tipp

«instanceData» ist in LO-Versionen bis einschließlich LO 7.3 nicht änderbar. Eine Umbenennung ist bis LO 7.4 nur über das Einlesen einer XML-Datei möglich:

```
001  <?xml version="1.0" encoding="UTF-8"?>
002  <Mitglied>
003  </Mitglied>
```

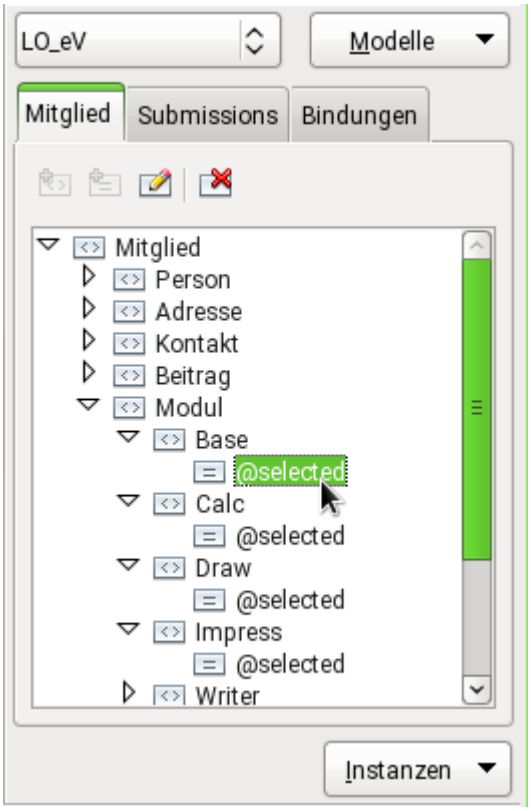
Wird der obige Inhalt als einfache Textdatei abgespeichert und mit dem Namen «Start.xml» versehen abgespeichert, so kann über **Instanzen → Bearbeiten** der Link zu dieser Datei eingelesen werden. «Mitglied» wird dann zum Startbegriff.
Nachdem «Mitglied» als Startbegriff erscheint sollte der Link wieder aus dem Dialog entfernt werden. Der Button Instanz verknüpfen ist hier ohne Funktion. 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.

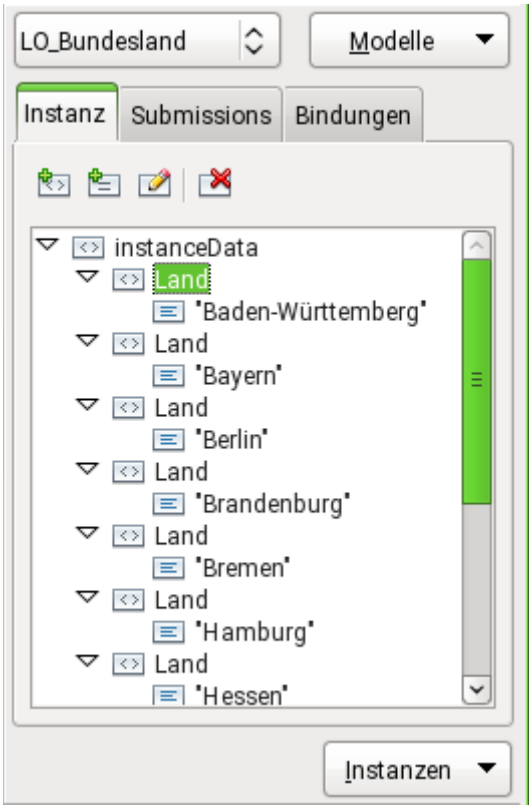
Werden den Feldern direkt auch über **Element bearbeiten → Einstellungen → Datentyp** Datentypen zugeordnet, so erstellt der Daten-Navigator automatisch die entsprechenden Bindungen, die für die Formularfelder nötig sind.

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.



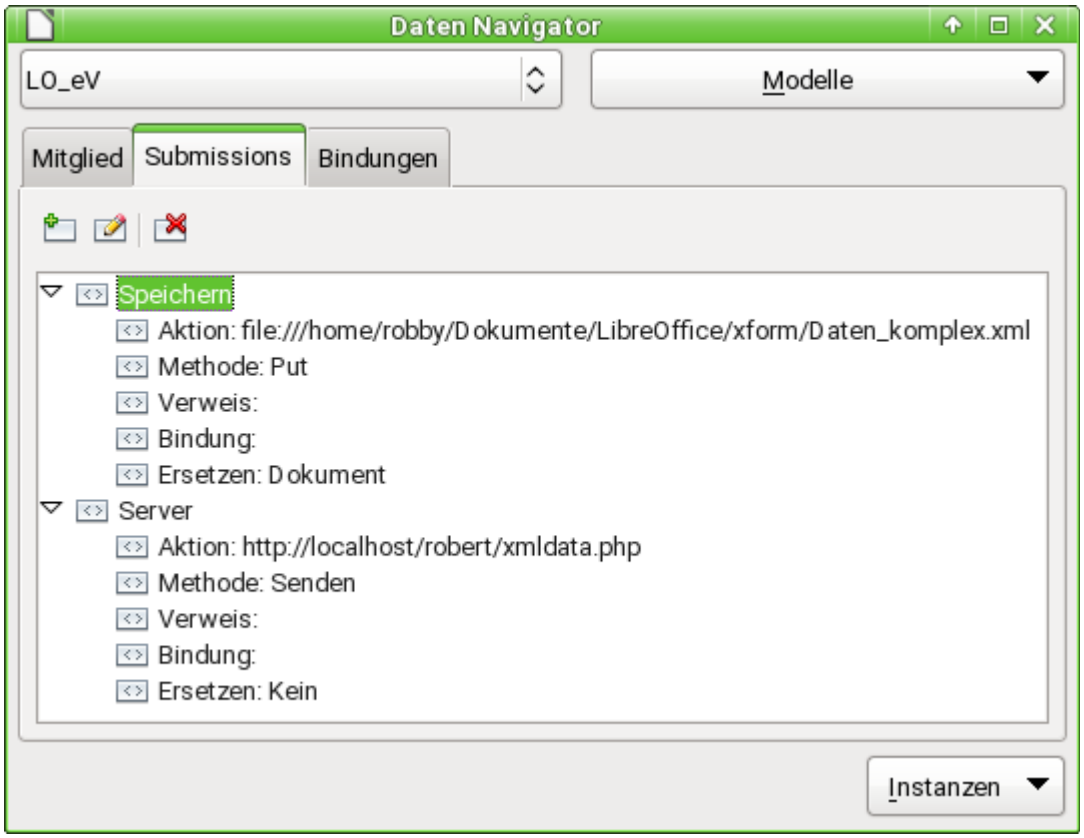
Für den Inhalt von Listenfeldern müssen die Werte in einem neuen Modell, hier «LO_Bundesland», erstellt werden. Dabei sind alle Elemente der 2. Stufe mit gleichem Namen benannt. So

kann später über die Bindungen eine Liste aller Felder abgefragt werden. Die entsprechenden Landesbezeichnungen über **Element bearbeiten →Element → Standardwert** 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.

Keine der Methoden dient dazu, nur einen Teil der Daten weiter zu bearbeiten. Dies ließe sich über den **Verweis** (**Bindungsausdruck**) oder die **Bindung** zu einem Feld bewerkstelligen.

Als **Bindungsausdruck** könnte z.B. über «/Mitglied/Person» nur der Teil weitergegeben werden, der unterhalb des Knotenpunktes «Person» abgespeichert ist.

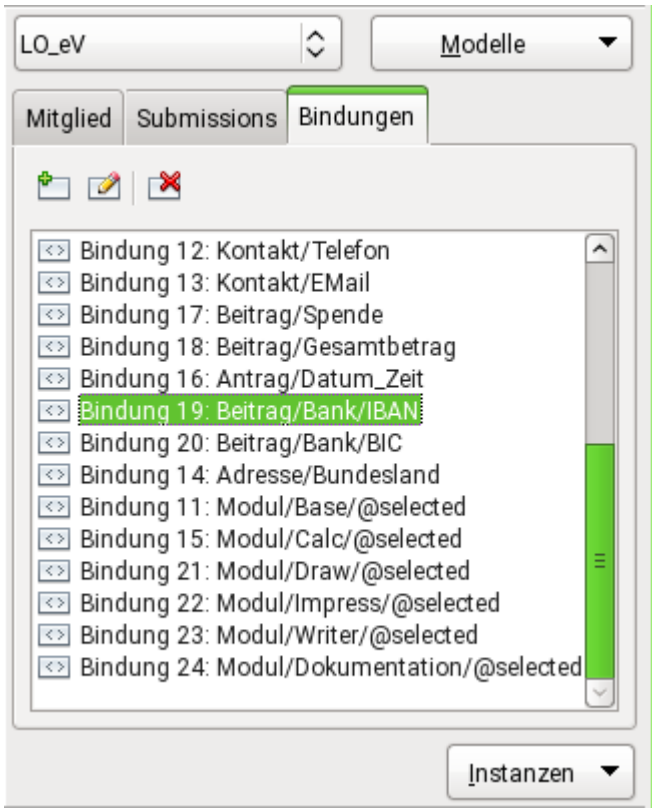
Für die **Bindung** an ein Feld ist allerdings Vorsicht geboten, da die Auswahl einer Bindung durch die GUI nicht mehr entfernt werden kann.

Tipp

Manche Eigenschaften der GUI lassen darauf schließen, dass die Formularerstellung noch nicht ganz ausgereift ist. Hier lässt sich nachsteuern, indem das Dokument geschlossen wird und anschließend mit einem Packprogramm testweise eine Kopie entpackt wird.
In der entpackten «content.xml» lässt sich mit einem einfachen Texteditor jetzt z.B. nach der fälschlicherweise erstellten Bindung suchen und dieser Eintrag löschen. Anschließend wird diese «content.xml» wieder in die Kopie eingelesen. Funktioniert alles wunschgemäß, so kann das Original durch die Kopie ersetzt werden.

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.



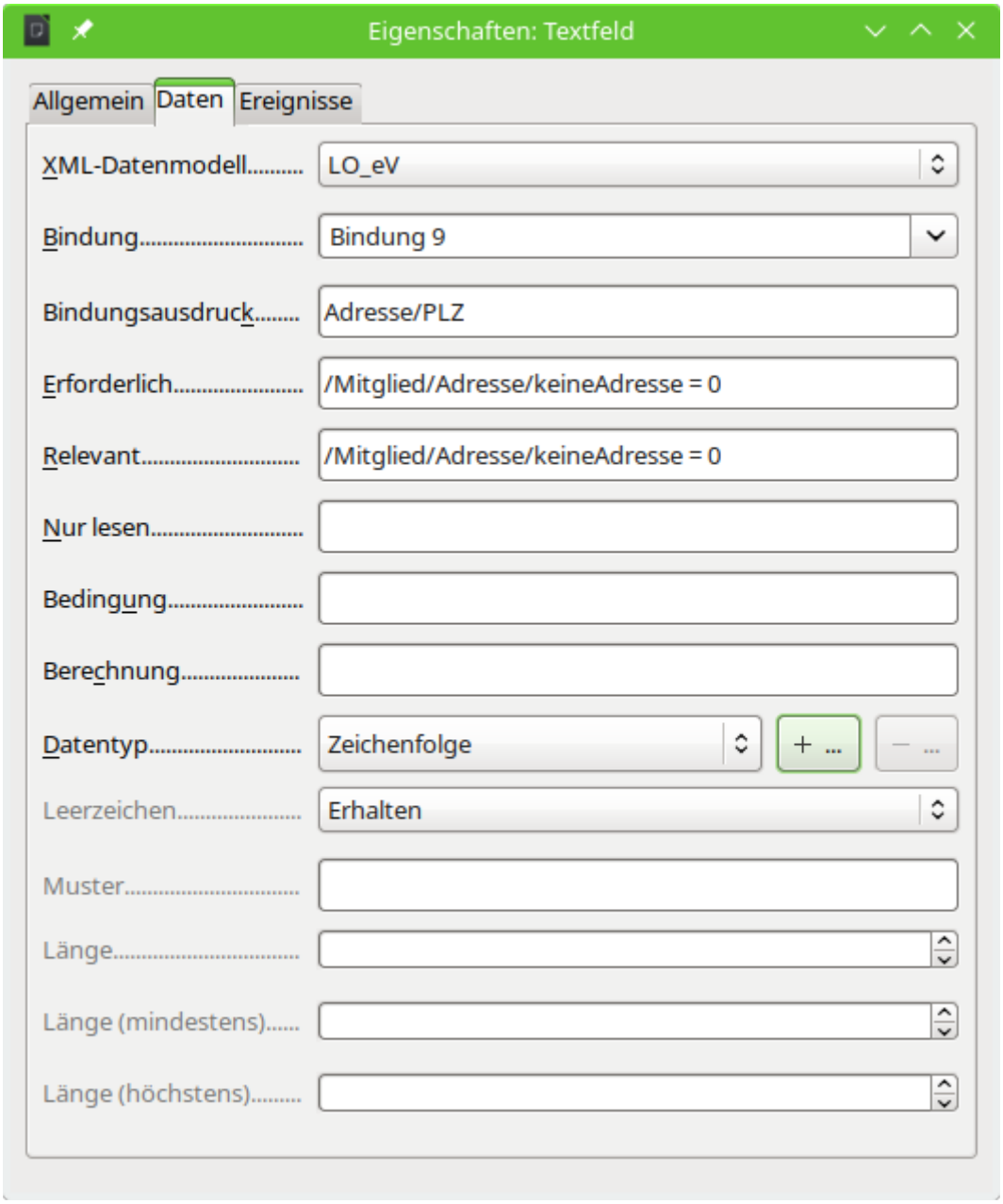
Bei der Bindungsbezeichnung muss darauf geachtet werden, dass die Bezeichnungen bei unterschiedlichen Modellen nicht gleichen. Die Namensgebung funktioniert nämlich nur innerhalb eines Modells einwandfrei. Bindungen werden dort einfach durchnummeriert. Wird eine Bindung gelöscht, so wird die Nummer anschließend 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. Hier sollte aber, wie bei Base-Formularen, besser im Reiter **Allgemein** der **Gruppenname** gleich gesetzt werden, da die Tabulatornavigation bei gleichen Feldnamen nicht korrekt verläuft.

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 9» das Feld «Adresse/PLZ» verbirgt?



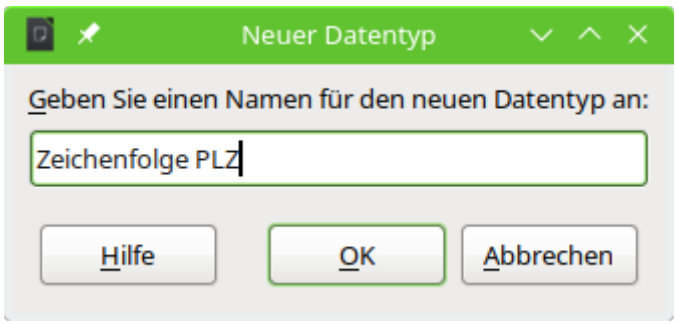
Die weiteren Einstellungen entsprechen dem der Elementdefinition.

Vorsicht

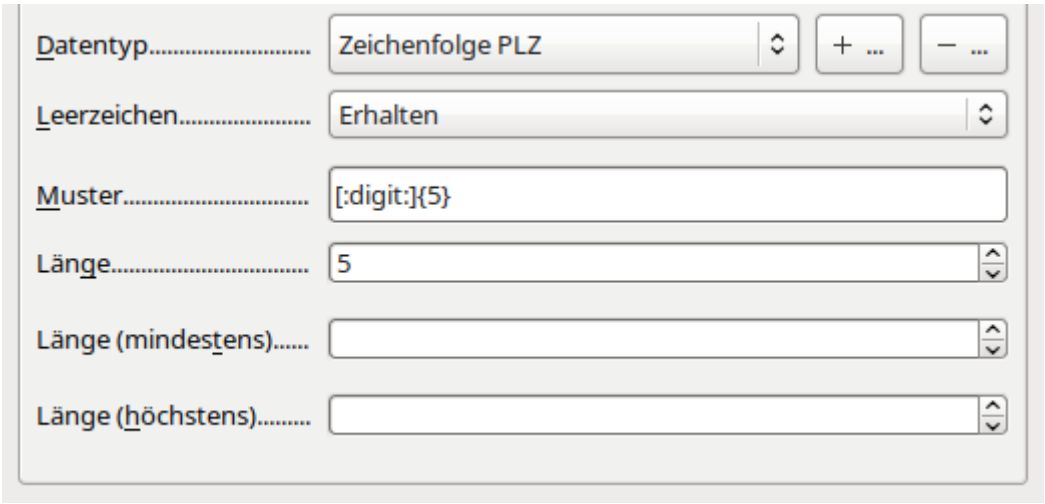
Die Eingaben unterhalb des Datentyps sind zwar ausgegraut. Sie lassen sich dennoch auch für voreingestellte Datentypen nutzen. Dadurch werden die Datentypen für alle entsprechenden Felder neu definiert.
Würde hier also z. B. für den Datentyp **Zeichenfolge** eine Länge von höchstens 10 Zeichen eingegeben, so würden alle Felder mit dem Datentyp **Zeichenfolge** maximal 10 Zeichen erlauben.

Hier ist allerdings das folgende Vorgehen zu beachten:

- Es sollte über **Datentyp → Zeichenfolge → + ...** 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.

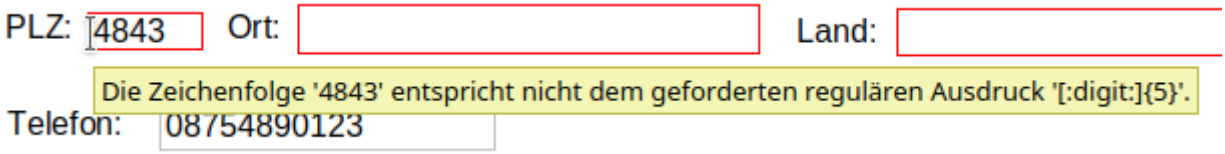


Ein neuer Datentyp wird erstellt, damit die Postleitzahl korrekt ist.

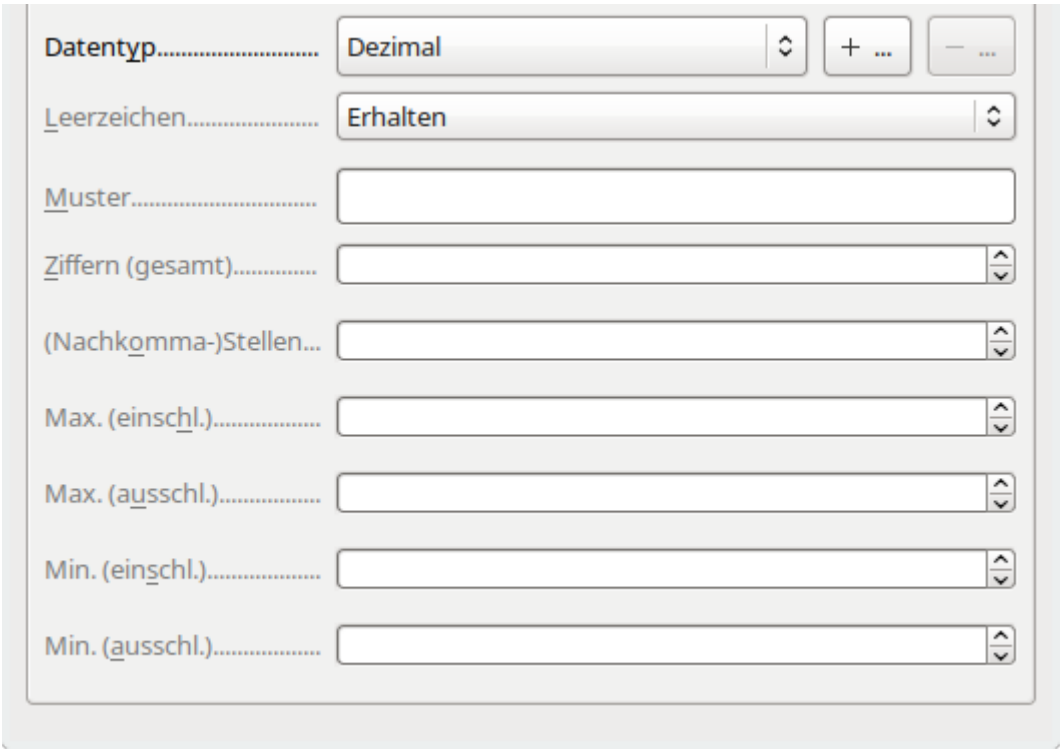


Die Elemente für den Datentyp «Zeichenkette PLZ» sind jetzt nicht mehr ausgegraut. Es ist jetzt ein Muster eingestellt, das grundsätzlich nur Zahlen zulässt. Es sollen insgesamt 5 Zahlen sein. Dies ist sowohl im Muster als auch im Längeneintrag eingestellt. Der Längeneintrag ist dabei gar nicht mehr notwendig.

Der obige Eintrag ist erst seit LO 7.6 möglich. Vorher wurde das Feld auch als fehlerhaft angezeigt, wenn kein Eintrag gemacht wurde. Deshalb mussten Einträge mit einer Untergrenze von 0 festgesetzt werden: **`[:digit:]{0,5}`**.



Grundsätzlich führt diese Eingabe nur dazu, dass das entsprechende Feld mit einer roten Umrandung und einer Fehlmeldung versehen wird. Wie bei allen anderen Datentypen kann die Eingabe trotzdem abgespeichert und weiter geleitet werden.



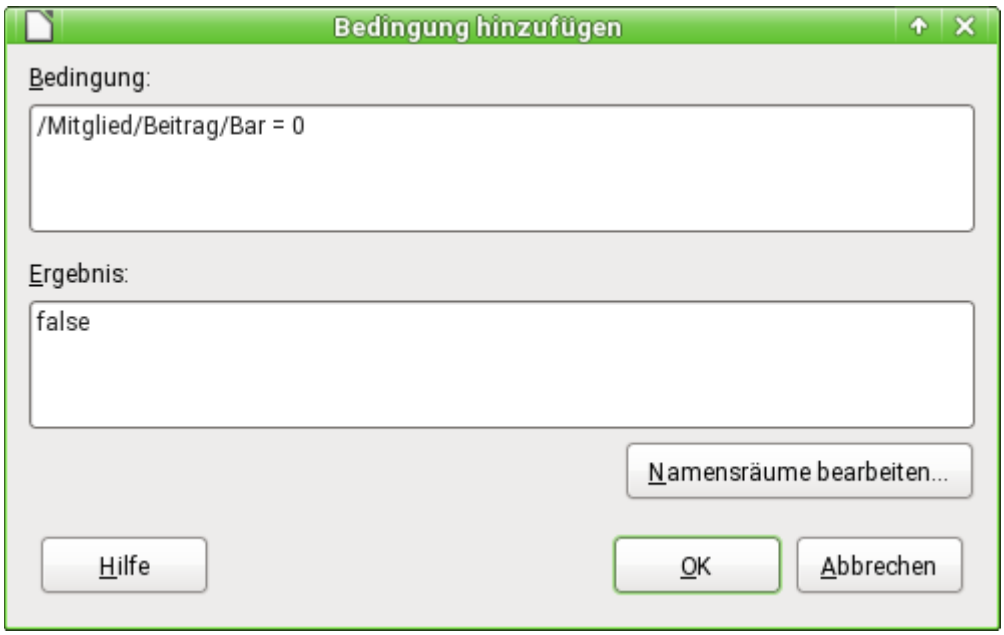
Für die Postleitzahl wurde die **Zeichenfolge** als Grundlage genommen. Beim Datentyp **`Dezimal`** sind die Einstellungsmöglichkeiten entsprechend auf Zahleneingaben zugeschnitten.

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.

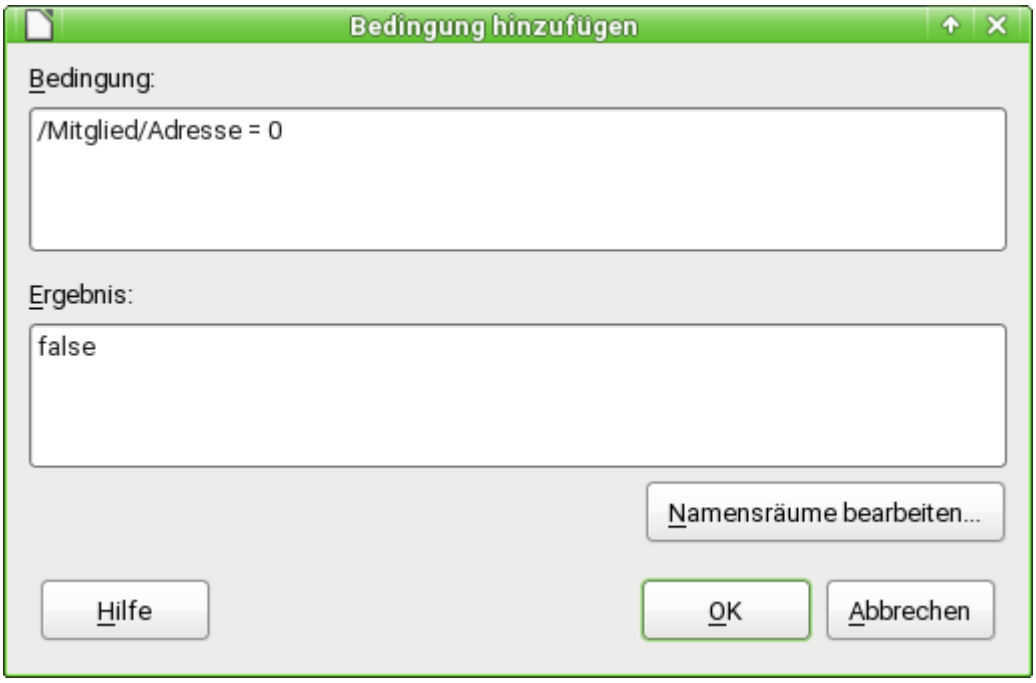


Hier wird eine Bedingung für das Feld «/Mitglied/Beitrag/Bank/IBAN» oder auch «/Mitglied/Beitrag/Bank/BIC» formuliert. Ist über das Feld «/Mitglied/Beitrag/Bar» keine Wahl erfolgt (Wert: 0), dann soll eine Eingabe erforderlich sein. Ansonsten hat sich die Eingabe in diesem Feld erledigt. Im Ergebnis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

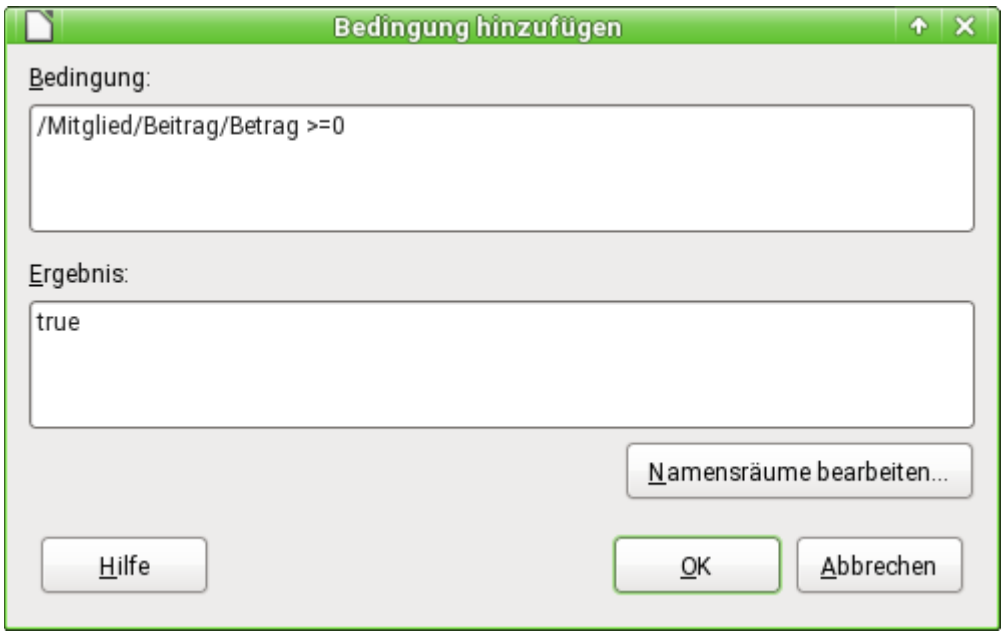
«keine Adresse angeben» ist nicht ausgewählt. Alle Felder für die Adresse erscheinen rot umrandet, da jetzt überall eine Eingabe erforderlich ist.

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

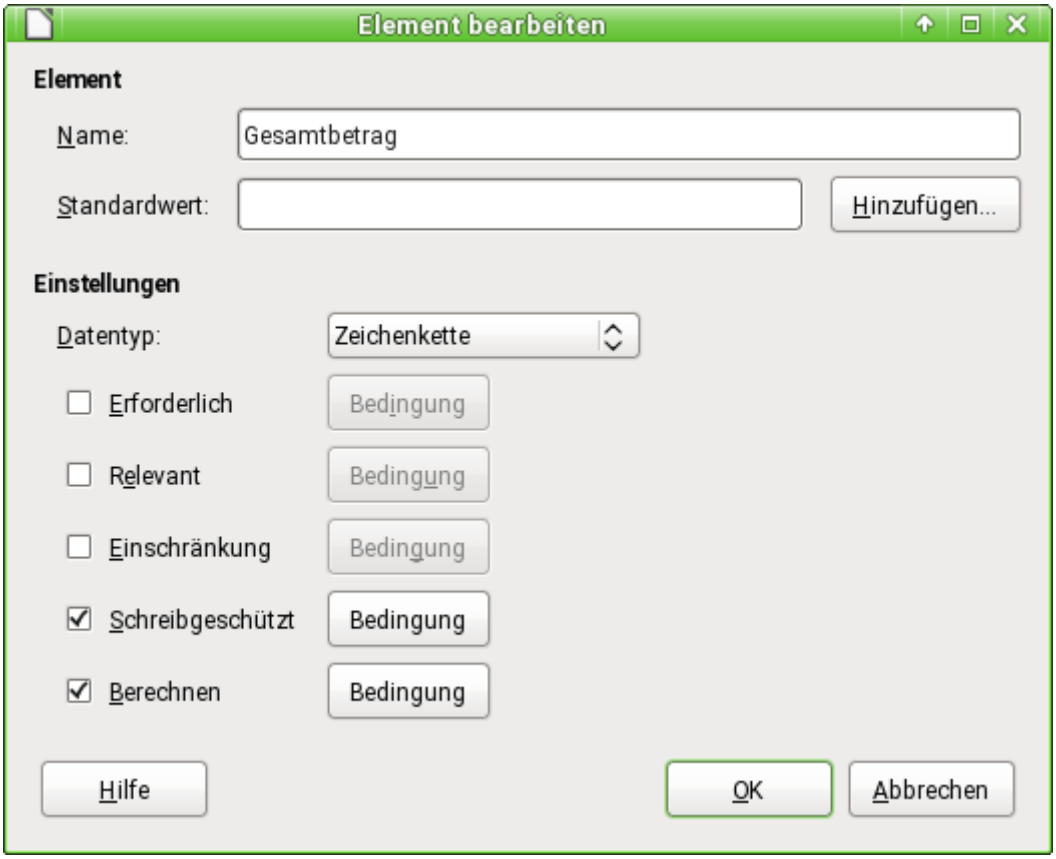
Beim «Betrag» und bei der «Spende» ist eine Einschränkung hinzugefügt worden:



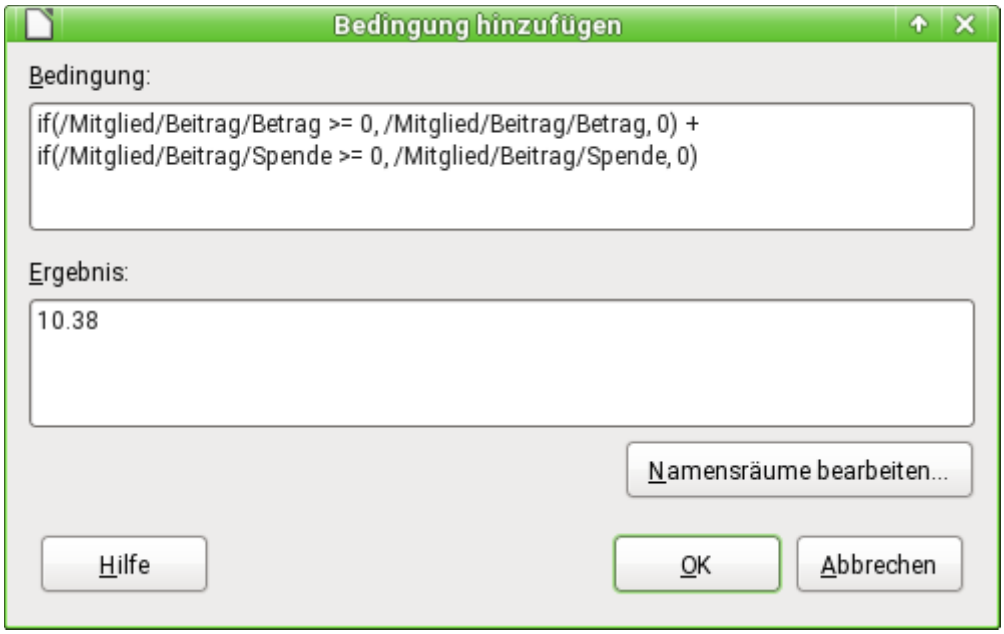
Es sollen schlicht nur Beträge erlaubt sein, die größer oder gleich 0 sind. Negative Zahlen werden also ausgeschlossen.

Schreibschutz und Berechnen

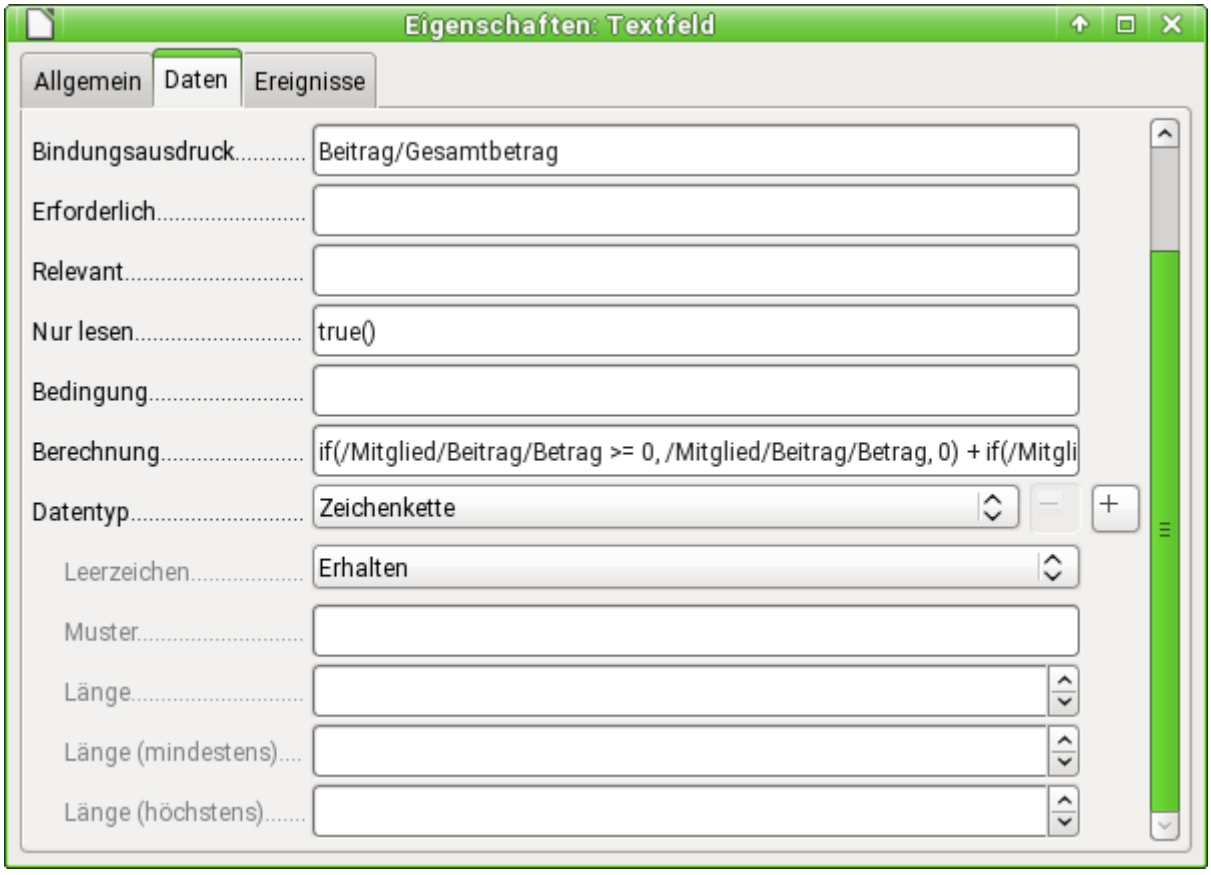
Für den Gesamtbetrag erfolgt eine Berechnung. Die Beträge sind hier in formatierbaren Feldern erfasst. In Versionen vor LibreOffice 7.6 funktionierte zwar diese Berechnung, nur wurde das Ergebnis nicht als Zahl korrekt ausgegeben. Dort war deshalb ein Textfeld statt eines formatierbaren Feldes für die Ausgabe notwendig.



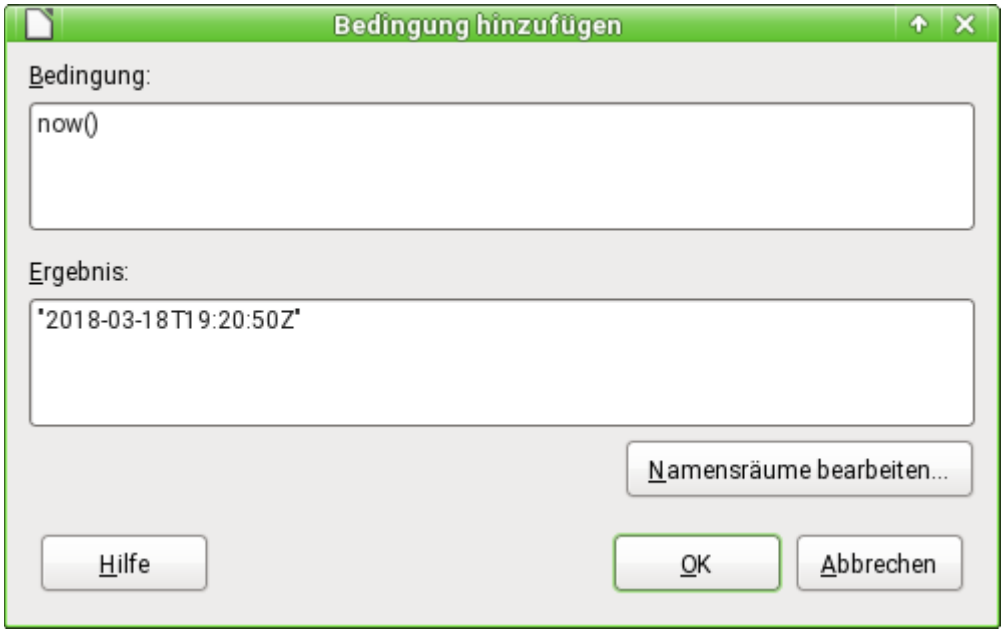
Die Berechnung erfolgt durch die Addition der Felder, die hier berücksichtigt werden sollen. Der Editor versteht hier **+**, **-**, ***** und **div** (nicht: «/», da bereits anderweitig belegt) als Rechenanweisungen. Enthält das jeweilige Feld keinen Eintrag, so muss für die Ermittlung der Summe stattdessen 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:

```
001 <tr>
002    <th></th>
003    <th>Betrag</th>
004    <th>Spende</th>
005    <th>Gesamt</th>
006    <th>Bar</th>
007    <th>IBAN</th>
008    <th>BIC</th>
009    </tr>
010 <tr>
011    <td></td>
012    <td><xsl:value-of select="Beitrag/Betrag"/></td>
013    <td><xsl:value-of select="Beitrag/Spende"/></td>
014    <td><xsl:value-of select="Beitrag/Gesamtbetrag"/></td>
015    <td><xsl:if test="Beitrag/Bar=1">Ja</xsl:if>
016       <xsl:if test="Beitrag/Bar=0">Nein</xsl:if></td>
017    <td><xsl:if test="Beitrag/Bank/IBAN=''"> -</xsl:if>
018       <xsl:value-of select="Beitrag/Bank/IBAN"/></td>
019    <td><xsl:if test="Beitrag/Bank/BIC=''"> -</xsl:if>
020       <xsl:value-of select="Beitrag/Bank/BIC"/></td>
021 </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.

```
001 <tr>
```

```
002     <th></th>
003     <th colspan="5">Module</th>
004     <th>Antragszeitstempel</th>
005  </tr>
006  <tr>
007     <td></td>
008     <td colspan="5">
009        <xsl:if test="Modul/Base/@selected='true'">Base </xsl:if>
010        <xsl:if test="Modul/Calc/@selected='true'">Calc </xsl:if>
011        <xsl:if test="Modul/Draw/@selected='true'">Draw </xsl:if>
012        <xsl:if test="Modul/Impress/@selected='true'">Impress </xsl:if>
013        <xsl:if test="Modul/Writer/@selected='true'">Writer </xsl:if>
014        <xsl:if test="Modul/Dokumentation/@selected='true'">Dokumentation </xsl:if>
015     </td>
016     <td><xsl:value-of select="Antrag/Datum_Zeit"/></td>
017  </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[9]

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.

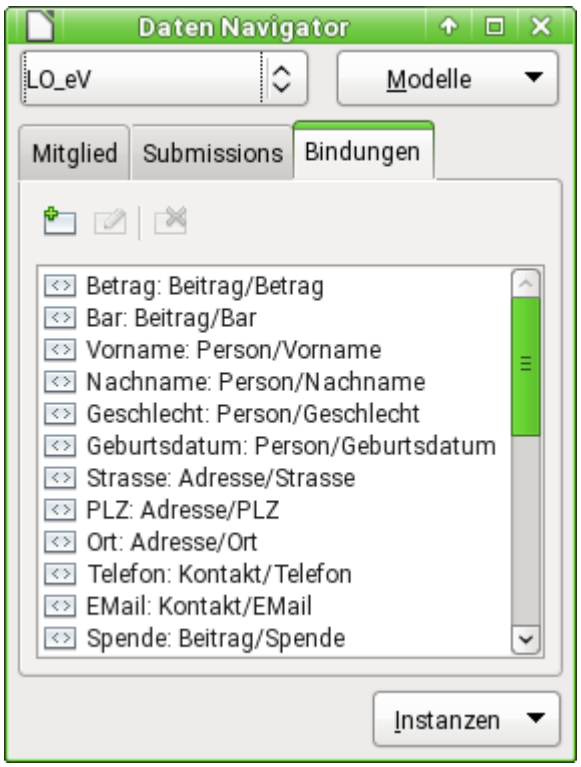
Mit Base besteht die Möglichkeit, Datensätze auszulesen und diese dann in den XML-Formularen darzustellen, zu verändern und neue Daten hinzuzufügen. Hierzu ist allerdings der Einsatz von Makros unabdingbar.

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:

9 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.

```
001  oBinding = thisComponent.XForms.getByName("LO_eV").Bindings
002  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:

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

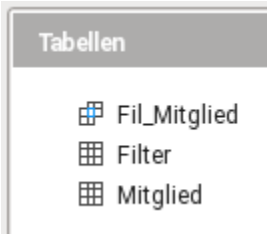
Bei numerischen Feldern gibt es die Methode **setEmpty** nicht. Hier werden die Felder mit **setText** zurückgesetzt:

```
001  oFieldBetrag = oForm.getByName("numBetrag")
002  oViewBet = oController.getControl(oFieldBetrag)
003  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:

```
001 SELECT * FROM "Mitglied" WHERE
002    IFNULL( LOWER ( "Nachname" ), '' ) LIKE '%' ||
003    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.[10]

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.

```
001 SUB DatenTabelle_Start
002    oXForm = ThisComponent
003    stDir = Left(oXForm.Location,Len(oXForm.Location)-Len(oXForm.Title))
004    stDir = ConvertToUrl(stDir & "XML_Daten.odb")
005    oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
006    oDatasource = oDatabaseContext.getByName(stDir)
007    oConnection = oDatasource.GetConnection("","")
008    oStatement = oConnection.createStatement()
```

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

```
009     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 beschreibbare Ansicht "Fil_Mitglied" zugegriffen werden. Für Datenänderungen wird später auf den direkten SQL-Code zurückgegriffen.

```
010     stQuery = "SELECT ""ID"", ""Vorname"", ... , ""Datum_Zeit""
           FROM ""Fil_Mitglied"""
011     oResult = oStatement.executeQuery(stQuery)
012     IF oResult.last THEN
013        stLast = oResult.getRow
014     ELSE
015        stLast = "0"
016     END IF
```

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

```
010 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.

```
017     IF oResult.first THEN
018        stPosition =  oResult.getRow
019        WerteInFormular
020     ELSE
021        stPosition = "0"
022        WerteInFormularReset
023     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.

```
024     oStatementF = oConnection.createStatement()
025     stSql = "SELECT ""Nachname"" FROM ""Filter"" WHERE ""ID"" = TRUE"
026     oResultF = oStatementF.executeQuery(stSql)
027     oResultF.Next
028     stFilter = oResultF.getString(1)
029     ThisComponent.Drawpage.Forms.getByIndex(0).
030        getByName("Textfeld_Filter_Nachname").Text = stFilter
031 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.

```
001 GLOBAL oConnection AS OBJECT
002 GLOBAL oStatement AS OBJECT
003 GLOBAL oResult AS OBJECT
004 GLOBAL stLast AS STRING
005 GLOBAL stPosition AS STRING
```

Die Navigationsleiste 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.

```
001 SUB ErsterDatensatz
002    IF oResult.First THEN
003       stPosition = oResult.getRow
004    ELSE
005       stPosition = "0"
006    END IF
007    WerteInFormular
008 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.

```
001 SUB VorherigerDatensatz
002    IF oResult.isFirst THEN
003       stPosition = oResult.getRow
004    ELSEIF stPosition > stLast THEN
005       oResult.absolute(stLast)
006       stPosition = stLast
007    ELSE
008       oResult.previous
009       stPosition = oResult.getRow
010    END IF
011    WerteInFormular
012 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.

```
001 SUB NaechsterDatensatz
002    IF oResult.isLast THEN
003       stPosition = oResult.getRow + 1
004       WerteInFormularReset
005    ELSE
006       oResult.next
007       stPosition = oResult.getRow
008       WerteInFormular
009    END IF
010 END SUB
```

```
001 SUB LetzterDatensatz
002    IF oResult.isLast THEN
003       stPosition = oResult.getRow + 1
004       WerteInFormularReset
005    ELSE
006       oResult.Last
007       stPosition = oResult.getRow
008       WerteInFormular
009    END IF
010 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.

```
001 SUB NeuerDatensatz
002    oResult.Last
003    stPosition = oResult.getRow + 1
004    WerteInFormularReset
005 END SUB
```

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

```
001 oBinding = thisComponent.XForms.getByName("LO_eV").Bindings
002 stID = oBinding.getByName("ID").getValue("string")
003 stVorname = String_to_SQL(oBinding.getByName("Vorname").getValue("string"))
004 ...
```

Existiert bereits ein Eintrag für das Feld «ID», so handelt es sich um einen bereits existierenden Datensatz. Folglich muss über SQL ein Update 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.

```
005 IF stID <> "" THEN
006    'Update
007    stSql = "UPDATE ""Mitglied"" SET ""Vorname"" = "+stVorname+",
          ""Nachname"" = "+stNachname+"," ...
008    stSql = stSql + " WHERE ""ID"" = "+stID+""
009    oStatement1 = oConnection.createStatement()
010    oStatement1.executeUpdate(stSql)
011 ELSE
012    'Insert
013    stSql = "INSERT INTO ""Mitglied"" (""Vorname"",""Nachname"",""," ... ")"
014    stSql = stSql + " VALUES ("+stVorname+","+stNachname+"," ... ")"
015    oStatement1 = oConnection.createStatement()
016    oStatement1.executeUpdate(stSql)
017    DatenTabelle_Start
018    oResult.Last
019    stPosition = oResult.getRow
020    WerteInFormular
021 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.

```
001 FUNCTION String_to_SQL(st AS STRING)
002    IF InStr(st,"'") THEN
003       st = Join(Split(st,"'"),"''")
004    END IF
005    IF st = "" THEN
006       st = "NULL"
007    ELSEIF st = "1" OR st = "0" THEN
```

```
008         st = st
009      ELSE
010         st = "'" & st & "'"
011      END IF
012      String_to_SQL = st
013   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.

```
001   FUNCTION XMLTime_to_SQLTime(st AS STRING)
002      st = Join(Split(st,"T")," ")
003      st = Join(Split(st,"Z"),"")
004      XMLTime_to_SQLTime = st
005   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 Tabelle "Filter" erhält ein Update und anschließend wird die Datentabelle neu eingelesen.

```
001   SUB Filtern(oEvent AS OBJECT)
002      oButton = oEvent.Source.Model
003      stFilter = oButton.Parent.getByName(oButton.Tag).Text
004      stSql = "UPDATE ""Filter"" SET ""Nachname"" = '"+stFilter+"'
            WHERE ""ID"" = TRUE"
005      oStatementF = oConnection.createStatement()
006      oStatementF.executeUpdate(stSql)
007      DatenTabelle_Start
008   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[11]

Werden über das Formular Daten als XML-Datei abgespeichert, so kann es zur Auswertung auf einem anderen Rechner auch sinnvoll sein, die Daten wieder einlesen zu können. Gegebenenfalls 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. folgenden Inhalt:

```
001   <?xml version="1.0" encoding="UTF-8"?>
002   <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><Spende>10</Spende><Gesamtbetrag>10</
      Gesamtbetrag><Bank><IBAN/><BIC/></Bank></Beitrag><Modul><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.

```
001   SUB Import
002      oDoc = ThisComponent
003      stDir = Left(oDoc.Location,Len(oDoc.Location)-Len(oDoc.Title))
004      stFile = stDir & "Beispiel_4.xml"
```

11 Siehe hierzu das Beispiel «Formular_komplex_Base_Import.odt»

```
005     oFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
006     IF oFileAccess.exists(stFile) THEN
007        oInputStream = createUnoService("com.sun.star.io.TextInputStream")
008        oFile = oFileAccess.OpenFileReadWrite(stFile)
009        oInputStream.SetInputStream(oFile.getInputStream)
010        DO WHILE NOT oInputStream.isEOF
011           stTmp = stTmp & oInputStream.ReadLine()
012        LOOP
013        oInputStream.closeInput
```

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

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

Die nachfolgenden Felder haben den folgenden Aufbau:

```
001 <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:

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

Dies führt zu folgender Trennung:

```
001 <Mitglied
002 Person
003 Vorname>Lara</Vorname
004 Nachname>Schmidtke</Nachname
005 Geschlecht>w</Geschlecht>
006 ...
007 Modul
008 Base selected="false"/>
009 ...
```

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.

```
016        k = 0
017        FOR i = 0 TO UBound(aTmpData)
018           IF InStr(aTmpData(i),"</") OR InStr(aTmpData(i),"=") THEN
019              REDIM PRESERVE aField(k)
020              REDIM PRESERVE aData(k)
021              IF InStr(aTmpData(i),"</") THEN
022                 b = split(aTmpData(i),">")
023                 c = split(b(1),"</")
024                 aField(k) = b(0)
025                 aData(k) = c(0)
026              END IF
027              IF Instr(aTmpData(i),"=") THEN
028                 b = split(aTmpData(i)," ")
029                 c = split(b(1),"""")
030                 aField(k) = b(0)
031                 aData(k) = c(1)
032              END IF
033              k = k+1
034           END IF
035        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.

```
036        WerteInFormularReset
037        oBinding = oDoc.XForms.getByName("LO_eV").Bindings
```

```
038         FOR i = 0 TO UBound(aField)
039            oBinding.getByName(aField(i)).setValue(aData(i))
040         NEXT i
041      END IF
042   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[12]

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[13]

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

12 Siehe *https://www.w3.org/MarkUp/Forms/2006/xforms-qr*
13 Siehe *https://www.w3.org/MarkUp/Forms/2006/xforms-qr*

Datentypen

Englische Bezeichnung	***Deutsche Übersetzung***	***Umfang***	***Ansicht im Textfeld***	***Bevorzugtes Formularfeld***
Date	Datum	Datum, auch führende Nullen	2023-05-31	31.05.23 (Datumsfeld)
Date and Time	Datum und Zeit	Datum und Zeit, auch führende Nullen	2023-05-31T10:25:00	31.05.23 10:25 (Formatiertes Feld)
Day	Tag	1 bis 31, auch führende Nullen	31	31 (Formatiertes Feld, auf 2 Stellen eingestellt)
Decimal	Dezimal	unbegrenzt	2.35	2,35 (Formatiertes Feld, hier können auch Maßeinheiten wie «€» hinzugefügt werden)
Double precision	Doppelte Genauigkeit	Wie Dezimal	2.356789	2,356789 (Formatiertes Feld)
Floating point	Fließkomma	Wie Dezimal	2.356789	2,356789 (Formatiertes Feld)
Hyperlink	Hyperlink	Start mit http, https, ftp …	http://libreoffice.org	http://libreoffice.org (Textfeld)
Month	Monat	1 bis 12, auch führende Nullen	5 oder 05	05 (Formatiertes Feld, auf 2 Stellen eingestellt)
String	Zeichenfolge		beliebig	beliebig (Textfeld)
Time	Zeit	00:00 bis 24:00	10:25:00	10:25 oder andere Formate (Formatiertes Feld)
True/False (Boolean)	Wahr/Falsch (Boolean)	1 oder 0	1	✓ (Markierfeld)
Year	Jahr	2023, auch führende Nullen	2023	2023 (Formatiertes Feld)

Das **Formatierte Feld** ist hier das Allroundfeld. Es lässt auch Zeiten wie 24:00:00 zu, beschränkt Zahlen nötigenfalls auf 2 Zeichen (Tag, Monat), kann Maßeinheiten wie den «€» hinzufügen usw. Lediglich beim Datum hat das Datumsfeld den Vorteil, dass es sich aufklappbar schalten lässt. Beim Wahr/Falsch-Feld ist dann das Markierfeld durch die Anklickbox besser geeignet.

Das Textfeld sollte nur für Texteingaben wie eine Zeichenfolge oder den Hyperlink genutzt werden. Lediglich die Eingabe von Zeiten ist noch von der Zeichennutzung her für deutschsprachige NutzerInnen einfach um zu setzen. Datumseingaben und Kombinationen von Datum und Zeit sind gewöhnungsbedürftig. Auch die Eingabe des Dezimaltrenners in Form eines Punktes dürfte häufig zu Fehlern führen.

LibreOffice
The Document Foundation

XML-Formulare

Formulare gestalten und auslesen

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.

7.6