AIIM BP-04-2010

# BEST PRACTICE

## CREATING PDF FORMS FOR THE CLINICAL DOCUMENT ARCHITECTURE

## AN IMPLEMENTATION GUIDE

Approval date:
June 1, 2010

AIIM BP-04-2010

An AIIM Best Practice
Guide Prepared by AIIM

Find, Control, and Optimize
Your Information

AIIM BP-04-2010

# Creating PDF Forms for the Clinical Document Architecture: An Implementation Guide

**An AIIM Best Practices Guide**

Prepared by AIIM – Association for Information and Image Management

**Approved: June 1, 2010**

AIIM BP-04-2010

1100 Wayne Avenue, Suite 1100
Silver Spring, MD 20910-5603 USA
Telephone: 301.587.8202
Fax: 301.597.2711
E-mail: aiim@aiim.org
Website: http://www.aiim.org
ISBN # 0-89258-424-6

Printed in the United States of America.

# Foreword

### *Contributors to the Creating PDF Forms for the Clinical Document Architecture: An Implementation Guide*

Adobe Systems Incorporated

AIIM

American Academy of Family Physicians

American Academy of Pediatrics

ASTM International

CapMed

Dak Systems Consulting

eClinicalWorks LLC

Generator LLC

Good Health Network

Intel Corporation

Massachusetts Medical Society

Medirex Systems Inc.

NextGen

Northrop-Grumman

Schering-Plough

Solventus

Voice of the Physician

YourTimeMatters.com

### *Supporters of the Creating PDF Forms for the Clinical Document Architecture: An Implementation Guide*

American Academy of Pediatrics

Cerner Corp.

Epocrates

Harvard University

MedCommons Inc.

MEDecisions

MinuteClinic

Northern Illinois Physicians for Connectivity

SureScripts

# Contents

# Introduction

## Objective

Develop a Fillable PDF Form based on the *Emergency Information Form for Children With Special Needs* as an example of a Clinical Care Document that supports the Clinical Document Architecture Schema based on the following constraints:

1. Use the *Emergency Information Form for Children With Special Needs* as the foundation for the form.
2. The appearance of the paper form will be maintained.
3. The form will be Static. There will not be any dynamic subforms or flow in the rendering of the view (see Item 2). Text will be limited to the real estate of the respective form field of the paper version.
4. The underlying data structure will support a subset of the CDA Schema.
5. Only data elements necessary to support the form fields will be included in the underlying data structure.
6. The form will be structured text.
7. The data entry method for specific complex elements (e.g. names, addresses, medications) will support the capture of specific child elements.

## Assumptions

This section assumes a basic understanding of Adobe LiveCycle Designer ES 8.2 and Adobe Acrobat 9 Professional or comparable software. These products were used to create the reference tools for the development of the sample. Implementers may use any comparable tool set.

This section also assumes some experience in designing XML based forms, including Data Binding and SOM Expressions.

Additionally, the reader should have some familiarity with the HL/7 Clinical Document Architecture (CDA) as well as the Continuity of Care Document (CCD) specification, a constraint of the CDA Schema that defines the encoding, structure and semantics of patient summary clinical data.

## Use Case

The following steps describe the use case supported by the Fillable PDF Form based on the *Emergency Information Form for Children With Special Needs.*

1. A child's guardian needs the "Emergency Information Form…" for a camp his child is attending.
2. The guardian contacts the primary care physician (e.g. a family doctor).
3. An initial CDA Level-2[1] compliant XML dataset is generated by a primary care physician's office.
4. The form is populated with the XML dataset.
5. The form is electronically transferred (email, flash drive, secure website) to the child's guardian.
6. The child's guardian enters contact information into the form.

[1] CDA Level-2 compliance provides templates for sections and allows for human-readable data that can still be machine parsed.

7. The child's guardian signs the form and delivers it to the camp personnel.

## Tools

- Adobe LiveCycle Designer ES 8.2
- Adobe Acrobat 9 Professional

## References

- CCD XML dataset (Emergency Info Sample.xml)
- CDA Release 2.0, Normative Edition - May, 2005
  http://healthinfo.med.dal.ca/hl7intro/CDA_R2_NormativeWebEdition/
- Scripting Object Model Expression Reference
  http://partners.adobe.com/public/developer/en/xml/som_2.0.pdf
- LiveCycle Designer Scripting Reference
  http://help.adobe.com/en_US/livecycle/8.2/lcdesigner_scripting_reference.pdf
- XML Forms Architecture (XFA) Specification, Version 2.8
  http://partners.adobe.com/public/developer/en/xml/xfa_spec_2_8.pdf
- Adobe XML Form Object Model Reference
  http://partners.adobe.com/public/developer/en/xml/Adobe_XML_Form_Object_Model_Reference.pdf

## Samples Files

| File | Description |
|---|---|
| Emergency Info Sample.xml | Sample file that can be imported to populate the form. This file can be edited to provide different examples of the form. The structure of this file is the basis for the dataset within the XFA form. Normally, this data would be generated by the primary care physician's office. It is provided here with fictitious data to allow the user to experiment with variations of the dataset and understand the bindings between the XML dataset and the PDF Form fields. |
| EIF Template.xml | A template that can be imported into the original PDF ('EIF No Template.pdf') to ensure default values are set and a minimal data set is present so a blank form can be edited through Acrobat or Reader.<br><br>This file differs from 'Emergency Info Sample.xml' by excluding instance-specific data. It includes the attributes required to correctly bind form fields to the dataset (e.g. For the 'telecom' element, the 'Use' attribute is configured to differentiate between Work and Home phone numbers but |

| | |
|---|---|
| | the phone numbers themselves are not included.) |
| EIF.pdf | 'EIF No Template.pdf' with 'EIF Template.xml' imported. |
| EIF No Template.pdf | The XFA form without any imports. |

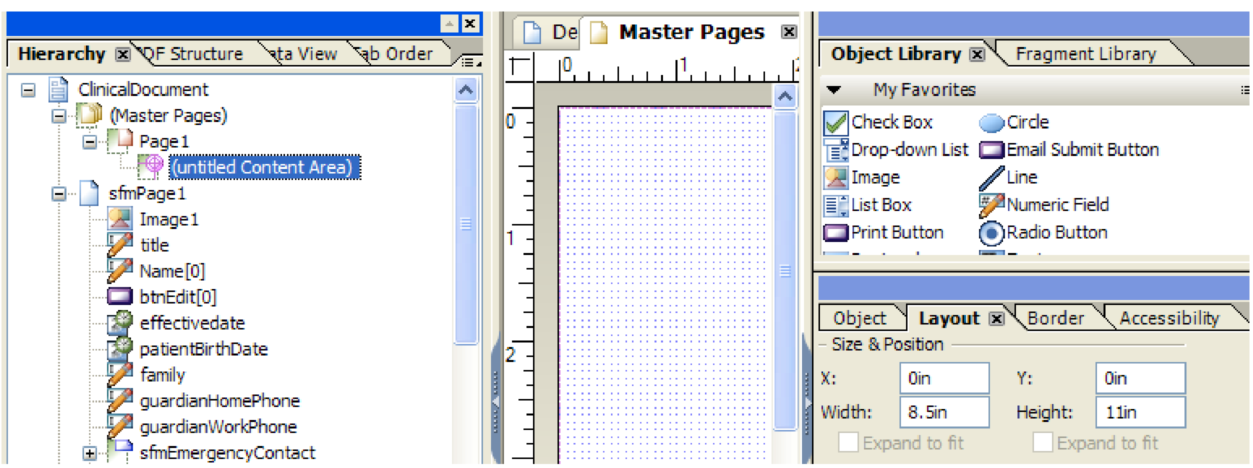
## Building the Form

### Form Background Graphics

Due to the desire to maintain the paper forms appearance, the basis of the fillable PDF form is a high resolution graphic from the scanned version of the Emergency *Information Form for Children With Special Needs* form. In Acrobat Designer, if a preexisting form is used as the basis for a Static form (maintains the appearance of the original form), subforms cannot be used. On the other hand, if a preexisting form is used as the basis for a Dynamic form (which supports subforms), the graphical elements are separated, and their placement does not necessarily match the original form.

In order to reproduce the *Emergency Information Form for Children With Special Needs form* accurately and still have subform support, a scanned image of each paper form page was used as the background for the respective page of the fillable PDF Form.

Since the image of each form page was 8.5" x 11", the size of the Content Area on the Master Page had to be changed to 8.5" x 11" as well. This is done by selecting the Content Area in the Hierarchy tab, then selecting the Layout tab and changing the dimensions appropriately as illustrated below:




### Data Binding

Data-enabled XFA-based PDF files can serve as a host for XML data. They provide import and export services for pre-configured XML instances or Schemas. In practice, this is accomplished by specifying a data connection that corresponds to a XML dataset representing the union of data elements required by the representative form as well as those required by the CCD constraints as applied to the CDA XML Schema.

Adobe LiveCycle Designer specifies these elements through the Data View panel. The following steps create a data element associated with the reference XML dataset:

1. Select menu item Window => Data View.
2. In the Data View menu (triangle at the top right), click on "New Data Connection".
3. Select "Sample XML Data", then name the Data Connection **EIF**, and click on "Next".
4. Select the XML file **Emergency Info Sample.xml**, which should be in the same directory. This should result in **Emergency Info Sample.xml** displayed in the edit box.
5. Under the Options dropdown "Use XML Data Root Element Name", select "ClinicalDocument".
6. Click "Finish".

This will create the appropriate ConnectionSet and Data Description Elements. In the "Data View" panel, you should now see a hierarchy of elements corresponding to the **Emergency Info Sample.xml** data file. An XML Dataset is used as the basis for the data connection because it provided a mechanism to constrain the fields available to those pertinent to the described workflow.

### Data Binding Fields and Subforms

Fields and subforms are bound by specifying the `<bind>` element. Attributes in the bind element include `match` and `ref`, where `match=` none, `once`, `dataref`, or `global`. The XML Forms Architecture (XFA) Specification (see References) specifies the default is `once`. The `ref` attribute specifies the explicit data node to match to in the case where `match=dataref`.

Adobe LiveCycle Designer specifies the `<bind>` element when the user drags an element from the "Data View" panel into the "Body Pages" section. Dragging an element into an empty area will result in the creation of a field, or a hierarchy with subforms and/or fields, depending on what element the user drags from the "Data View" panel. Dragging an element onto an existing field will result in the updating of the target field's bind element. The sample form's simpler bindings were specified using this mechanism.

### Tables

The sample makes extensive use of tables to represent the human-readable portion of the CDA-compliant document. XFA defines tables by using the `<subform>` elements with a layout property of "table".

<subform layout="table">

XFA's Layout engine will ensure that the layout of the specified fields conform to a tabular format. Authors can create tables in Designer by going to the Library panel and dragging a Table object to the Body page. Designer will then give the author options on creating the base table, which include specifying the number of columns and rows, and whether to include header and footer rows. Tables are excellent mechanisms for representing repeating data elements in XML by mapping the bindings in the Object Panel and Binding subpanel.

A table may also be created by dragging and dropping an element from the Data View panel onto the form. If the element has a large number of subelements or the subelements are very complex, the drag and drop method may require deleting many elements manually. In this circumstance, it may be easier to draw the table and manually define the bindings as previously described.

XFA defines Table Rows by using `<subform>` elements with a layout property of `"row"`. These rows can also specify bindings and can contain other XFA elements such as subforms, fields, and even other tables. Each XFA element contained in a row subform is considered a cell in the table.

### CDA Tabular Data Representation

The CDA Schema also provides mechanisms for representing tabular data. An excellent example of data in tabular form is the Medications section. According to the CDA Schema, the tabular data can be represented as text and formatted using a subset of markup language similar to HTML. Within this text block, it is possible to represent data as a table:

```
<title>Medications</title>
<text>
    <table border="1" width="100%">
        <thead>
            <tr>
                <th>Medication</th>
                <th>Instructions</th>
                <th>Start Date</th>
                <th>Status</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Albuterol inhalant</td>
                <td>2 puffs QID PRN wheezing</td>
                <td> </td>
                <td>Active</td>
            </tr>
            <tr>
                <td>Clopidogrel (Plavix)</td>
                <td>75mg PO daily</td>
                <td> </td>
                <td>Active</td>
            </tr>
            <tr>
                <td>Metoprolol</td>
                <td>25mg PO BID</td>
                <td> </td>
                <td>Active</td>
            </tr>
            <tr>
                <td>Prednisone</td>
                <td>20mg PO daily</td>
                <td>Mar 28, 2000</td>
                <td>Active</td>
            </tr>
            <tr>
                <td>Cephalexin (Keflex)</td>
                <td>500mg PO QID x 7 days (for bronchitis)</td>
                <td>Mar 28, 2000</td>
                <td>No longer active</td>
            </tr>
        </tbody>
```

```
    </table>
</text>
```

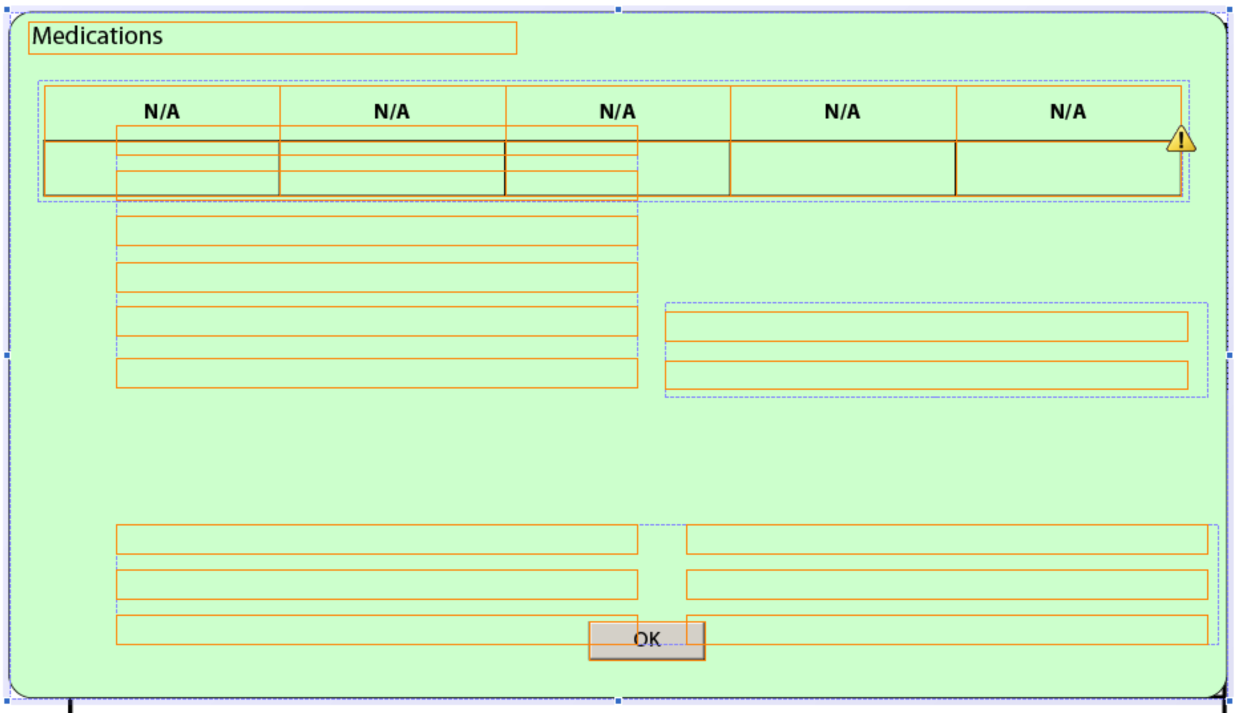
alternatively, the CDA Schema allows the same data to be represented as a list:

```
<title>Medications</title>
<text>
<list listType="unordered">
        <item>
                <content ID="med1">
                Medication: Albuterol inhalant<br/>
                Instructions: 2 puffs QID PRN wheezing<br/>
                Status: Active
                </content>
        </item>
        <item>
                <content ID="med2">
                Medication: Clopidogrel (Plavix)<br/>
                Instructions: 75mg PO daily<br/>
                Status: Active
                </content>
        </item>
        <item>
                <content ID="med3">
                Medication: Metoprolol<br/>
                Instructions: 25mg PO BID<br/>
                Status: Active
                </content>
        </item>
        <item>
                <content ID="med4">
                Medication: Prednisone<br/>
                Instructions: 20mg PO daily<br/>
                Start Date: Mar 28, 2000<br/>
                Status: Active
                </content>
        </item>
        <item>
                <content ID="med5">
                Medication: Cephalexin (Keflex)<br/>
                Instructions: 500mg PO QID x 7 days (for bronchitis)<br/>
                Start Date: Mar 28, 2000<br/>
                Status: No longer active
                </content>
        </item>
</list>
</text>
```

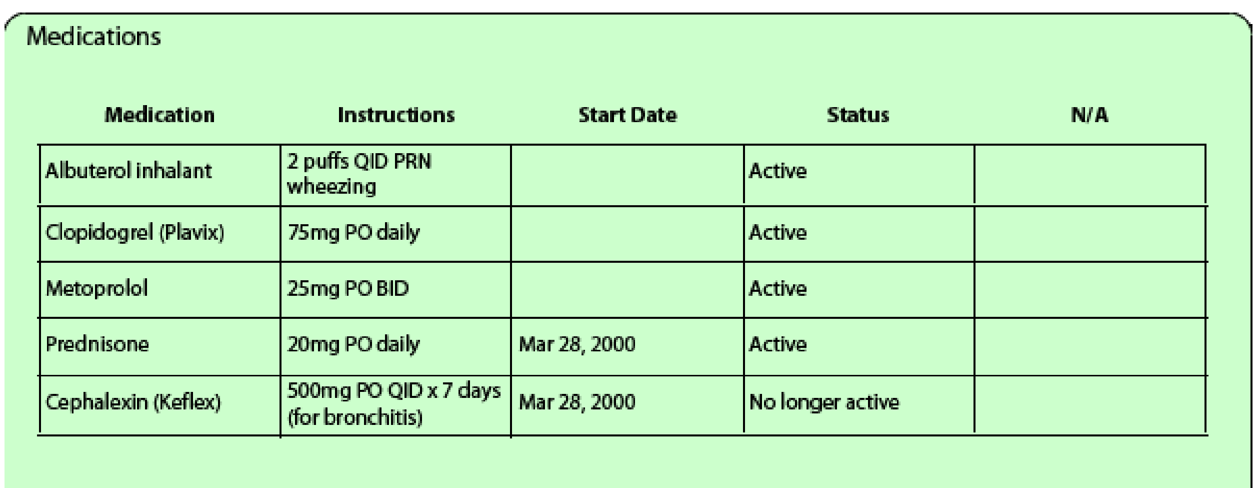
## Using Hidden Subforms to Parse the Dataset

Data in these sections are represented as markup and formatted as tables or lists. The specific attributes and properties in the dataset are mapped to cells in a table on a subform which allows for a one-to-one correspondence. This provides a simple method of concatenating groups of elements into a single string for presentation on the visible form.

The medications data in the XML dataset is mapped to the table displayed below on the Hidden subform, “sfmSectionTable”.




The columns map to properties of a single medication, while each row represents a different medication. When the form is populated from a compliant XML data file, the medications listed in the data file fill the individual cells of the hidden subform as illustrated below:



Medications

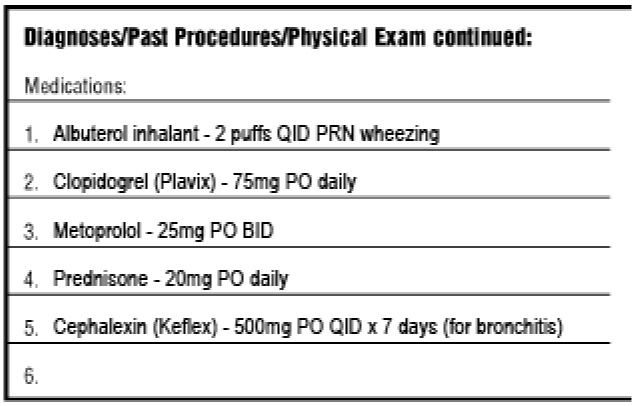
| Medication | Instructions | Start Date | Status | N/A |
|---|---|---|---|---|
| Albuterol inhalant | 2 puffs QID PRN wheezing | | Active | |
| Clopidogrel (Plavix) | 75mg PO daily | | Active | |
| Metoprolol | 25mg PO BID | | Active | |
| Prednisone | 20mg PO daily | Mar 28, 2000 | Active | |
| Cephalexin (Keflex) | 500mg PO QID x 7 days (for bronchitis) | Mar 28, 2000 | No longer active | |

In the instance above, the last column is not populated because the data file only included four properties for each medication. In other use cases, this subform could be used as a popup dialog allowing information to be entered by the form user.

Once the hidden subform table is populated from the dataset, the following JavaScript traverses the populated table and concatenates predetermined columns into a single string for display on the actual form (for added context, see the “MedsCode” module in the sample).

```
function SimpleSetMedicationsFields(strMeds)
{
     //Medications table body row of hidden subform
     var strBodyRows = "ClinicalDocument.sfmPage2.sfmSectionTable.table.tbody.tr";
     //Form medications field array base
     var strMedsFields = "ClinicalDocument.sfmPage2.frmMedications.Medication";
     //Number of form field array members (TODO: calculate from base element)
     var intListLen = 6;

     SimpleSetSectionTableFields(strBodyRows, strMedsFields, intListLen);

}


function SimpleSetSectionTableFields(sfmBodyRows, strFormField, intListLen)
{
     // Generic routine to read a table structure and concatenate specific columns
     // into a string to populate an array of text fields.  NOTE: Indices for
     // text field array correspond to table row numbers.

     /*  Parameters
     /    sfmBodyRows = SOM Path to the row element
     /    strFormField = SOM Path to the field array without an index
     /    intListLen = Number of items to be displayed on the form (text field array count)
     */

     // List of columns to include in concatenated string
     // TODO - determine this by searching through a terms list
     var arrColList = new Array(0,1);

     strRow = sfmBodyRows + "[*].#subform";
     tblRows= xfa.resolveNodes(strRow);
     if(tblRows) {
          intColCnt = arrColList.length;

          var intRowCnt = ((tblRows.length > intListLen) ? intListLen : tblRows.length);
          if (intRowCnt > 0) {
               for (i=0; i<intRowCnt; i++) {
                    // For each data row

                    // Get reference to the corresponding form field
                    var strFld = strFormField + "[" + i + "]";
                    var oFld = xfa.resolveNode(strFld);

                    if(oFld) {

                         // Array to hold field parts from columns
                         var strParts = new Array(intColCnt);

                         // Access a field in a repeating subform by looping through the nodes
                         for (j=0;j<intColCnt;j++) {
                              strColFld = "td[" + j + "]";
                              strParts[j] = tblRows.item(i).resolveNode(strColFld).rawValue;
                         }
                         var curFld = strParts.join(" - ");
                         oFld.rawValue = curFld;
                    }
               }
          }
     }
}
```

The JavaScript results in the following data display on the form:



**Diagnoses/Past Procedures/Physical Exam continued:**

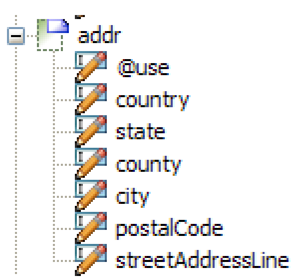
Medications:

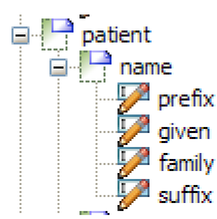
1. Albuterol inhalant - 2 puffs QID PRN wheezing
2. Clopidogrel (Plavix) - 75mg PO daily
3. Metoprolol - 25mg PO BID
4. Prednisone - 20mg PO daily
5. Cephalexin (Keflex) - 500mg PO QID x 7 days (for bronchitis)
6.

## Using Hidden Subforms as Popup Dialogs

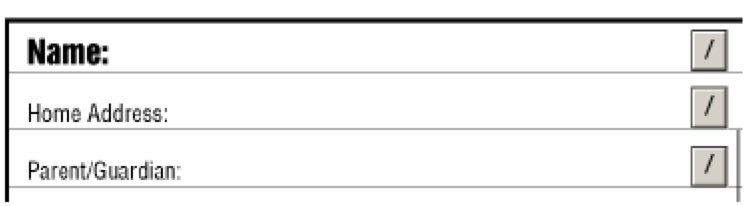
In this use case, the parent or guardian populates some contact information after the form has been populated with the XML dataset generated by the primary care physician. This contact information is stored in parts:

Address dataset Name dataset




To capture the contact information at the atomic level, these structures are mapped to fields on subforms. The subforms are hidden until the form user clicks on the edit button positioned at the right end of the fields:

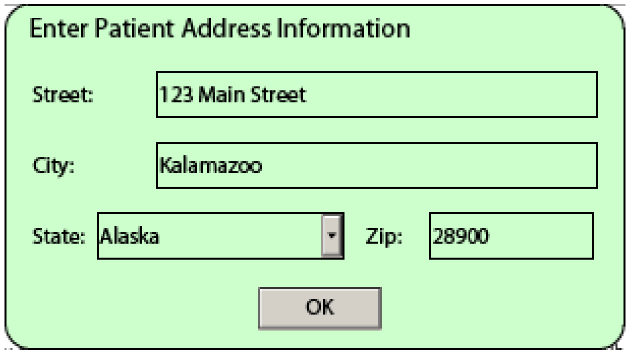


**Name:**

Home Address:

Parent/Guardian:

Once clicked, the buttons run a JavaScript that changes the respective subform's presence property from *Hidden* to *Visible*, which causes the subform to "popup" allowing the form user to enter the address information.




Once the user enters the data and clicks the OK button, the data is parsed and concatenated by the following JavaScript, which then populates the associated text fields on the form. The single parameter for the function, *intIndex*, links the popup subform to the visible subform that contains the destination text fields.

```
function SetAddrField(intIndex)
{
     // Set a text field value to a string concatenated from the atomic components in the
     // XML data set.

     // reference to subform containg the atomic-level fields
     var subName = xfa.form.resolveNode("ClinicalDocument.#subform[0].sfmName[" + intIndex +
"]");
     // reference to the destination field for concatenated string
     var fldName = xfa.form.resolveNode("ClinicalDocument.#subform[0].Name[" + intIndex + "]");

     // Concatenate string from atomic-level fields
     var strName = ((subName.streetAddressLine.rawValue!=null) ?
subName.streetAddressLine.rawValue + "\n" : "") +
     ((subName.city.rawValue!=null) ? subName.city.rawValue + ", " : "") +
     ((subName.state.rawValue!=null) ? subName.state.rawValue + "  " : "") +
     ((subName.postalCode.rawValue!=null) ? subName.postalCode.rawValue : "");

     // Assign concatenated string to destination field
     fldName.rawValue = strName;

}
```

The resulting form text fields: