



# Using XML Schema Definitions with Adobe® LiveCycle™ Designer 7.0

## TABLE OF CONTENTS

- 1 Introduction
- 2 XML Schema Definitions and LiveCycle Designer Software
- 3 XML Schema Definition Features Supported in LiveCycle Designer
- 3 Mapping XML Schema Definition Elements to Forms
- 4 Deriving Simple Types
- 5 Working with Complex Types
- 8 Understanding the Effects of Element Specifications
- 10 Using Validation Scripts
- 11 Getting Started with LiveCycle Designer
- 12 Conclusion

## Introduction

This paper describes how to use XML Schema definitions, written in XML Schema Definition Language (XSDL), to create electronic forms with Adobe® LiveCycle™ Designer 7.0 software. Creating electronic forms using XML Schema definitions:

- Moves responsibility for data definition from form designers to data designers
- Enforces consistency in electronic forms that depend on the same Schema definitions
- Simplifies and shortens the form design process

The paper is intended for:

- IT professionals who need to understand why and how using XML Schema definitions with LiveCycle Designer improves the form design process
- Schema designers who need to know how Schema definition elements are used in electronic forms
- Form developers who want to create forms by using XML Schema definitions with LiveCycle Designer

This paper assumes that you have some familiarity with XML, XML-related technologies, and the Adobe Intelligent Document Platform. You should also have a working knowledge of LiveCycle Designer, XSDL, and XML Forms Architecture (XFA).

## XML Schema definitions and LiveCycle Designer software

### Overview

LiveCycle Designer is used to create electronic forms. In the context of the Adobe Intelligent Document Platform, electronic forms are intelligent documents, containing presentation, data, metadata, and business logic. Intelligent documents provide the accessibility and readability of a traditional document while making it possible for end users to interact with business processes, all without custom programming.

If you have an XML Schema definition that describes the data your electronic form will generate, or the XML Schema definition contains data with which you want your form to be populated, LiveCycle Designer can use information about the elements in that definition to define the form's building blocks.

Figure 1 depicts how to use XML Schema definitions with LiveCycle Designer to define or generate a form.

The process begins when you establish a Data Connection between a Schema definition and LiveCycle Designer. Binding then relates elements in the XML Schema definition to fields and subforms in your electronic form. Binding allows a form's fields and subforms to automatically acquire various characteristics from XML Schema elements, including data type and validation rules. The form developer can adapt the default behaviors of XML Schema elements and customize presentation information. Generating the electronic form as a PDF document completes the process.

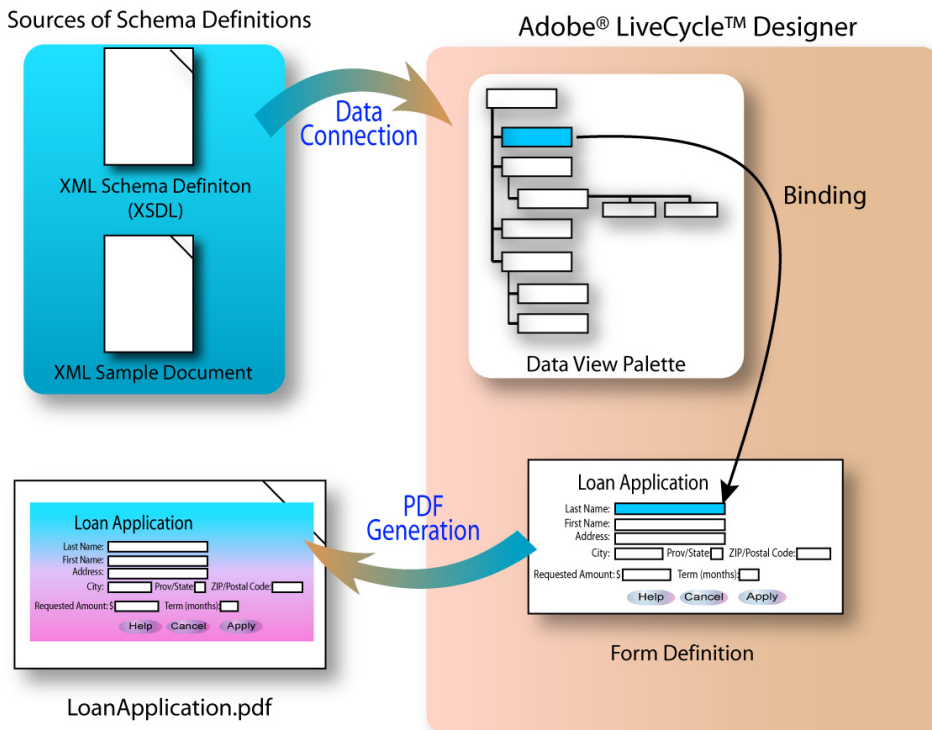


Figure 1: Using XML Schema definitions with LiveCycle Designer 7.0

**Note:** If you don't have an XML Schema definition for the data in your electronic form, you can use a sample XML document instead. LiveCycle Designer will generate a default form based on the apparent structure of the sample document. Although this method imposes fewer constraints on the structure of the XML that is derived from the form (because few documents exemplify a thorough Schema definition), it is still more efficient than designing a form from scratch.

## XML Schema definition features supported in LiveCycle Designer

LiveCycle Designer uses the Expat 1.95.7 XML parser to handle incoming XML and XML Schemas. In addition to the Expat libraries, LiveCycle Designer includes handlers written to map parse streams to the XFA language. LiveCycle Designer reads a Schema definition into an XML Document Object Model (DOM) as an XML file. The XML Schema is interpreted, compiled into a dataDescription, and ultimately saved to either an XML form data package (XDP) or PDF file. At form runtime, the dataDescription is used to guide the creation of data hierarchies to ensure that they conform to the original XML Schema definition.

This build of the Expat XML parser is non-validating. As a result, form developers need to ensure that XML and XML Schema rules are deterministically enforced. For more information on the Expat parser, see <http://expat.sourceforge.net>.

The Expat XML parser supports the following XML Schema mechanisms:

- Global and local element and attribute declarations
- Simple and complex types, including restrictions and extensions
- Default and fixed values for elements and attributes
- Sequence, choice, and all
- Reusable groups (<xsd:group> and <xsd:attributeGroup>)
- Annotation information
- Namespaces (use of targetNamespace attribute)
- The use of <xsd:include> for XML Schema composition
- The use of <xsd:import> for XML Schema composition
- The substitution group substitutionGroup and related attributes
- Support for XML names using the dot character

The Expat parser does not support the following XML Schema features:

- Non-native attributes
- The identity constraints <xsd:unique>, <xsd:key>, and <xsd:keyref>
- Redefinition using <xsd:redefine>
- The use of <xsd:notation>

## Mapping XML Schema definition elements to forms

### Overview

In LiveCycle Designer, forms are documents created from a hierarchy of building blocks known as subforms. Each subform controls a portion of the form's overall structure, presentation, and behavior. Individual subforms can contain a combination of objects that produce fillable regions (fields) or non-fillable regions (draws). Subforms can also contain other subforms, each with properties that determine how and when subforms are assembled into the form's final design.

In the XML Schema type hierarchy, there are simple types and complex types. Simple types are those with no subelements or attributes. Complex types may (but need not) contain subelements or attributes. In general, simple types map to a form's fields, and complex types usually map to subforms.

### Supported names for fields and subforms

LiveCycle Designer names fields and subforms after the XML Schema definition elements that map to them. Similarly, the initial caption for a field or subform is the name of the XML Schema element that maps to it.

With one exception, LiveCycle Designer 7.0 supports any element or attribute names in the source XML Schema definition (see Figure 1) that correspond to the XML 1.0 definition of an XML Name. The exception is that colons (":") can be used only as a namespace prefix separator.

**Note:** LiveCycle Designer 6.0 and Adobe Acrobat® Professional and Standard 6.0 do not support the use of dots (".") in element or attribute names. As a result, Acrobat Reader 6.0 can not open forms created in LiveCycle Designer 7.0 that have dots in element or attribute names.

**Object types in forms**

The W3C XML Schema specification defines over 40 built-in simple types. When element binding occurs, each simple type in the XML Schema definition creates an associated object type in the form. Table 1 shows this relationship for all of the default simple types.

SIMPLE TYPE, XML SCHEMA DEFINITION	OBJECT TYPE, FORM
<b>String and Name</b>	<b>String and Name</b>
language, Name, NCName, normalizedString, QName, string, token	TextEdit field
<b>Numeric</b>	<b>Numeric</b>
float	Numeric field - float data format
double	Numeric field - float data format
decimal	Numeric field - decimal data format
integer, long, int, short, byte, positiveInteger, nonPositiveInteger, negativeInteger, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte	Numeric field – integer format
<b>Date and Time</b>	<b>Date and Time</b>
duration	TextEdit
dateTime	Date/Time field of type dateTime
date	Date/Time field of type date
time	Date/Time field of type date
gYear, gYearMonth, gMonth, gMonthDay, gDay	TextEdit field with picture clause
<b>Legacy</b>	<b>Legacy</b>
ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, NOTATION	TextEdit field
<b>Other</b>	<b>Other</b>
boolean	Checkbox field
hexBinary	Image if recognize content type, otherwise TextEdit field
base64Binary	Image if recognize content type, otherwise TextEdit field
anyURI	TextEdit field

Table 1: Default object types created in forms by XML element binding

**Deriving simple types**

**Derivation by restriction**

You can derive new simple types from other simple types in several ways, including the use of restrictions. Such restrictions are expressed as constraining facets.

**Enumeration facets**

An element or attribute declaration of any base type that declares one or more enumerated values generates a pull-down list object in a form rather than the default object type for the declaration’s base type. The following sample code imposes this kind of restriction:

```
<xsd:element name="Colors">
  <xsd:simpleType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:enumeration value="red"/>
        <xsd:enumeration value="green"/>
        <xsd:enumeration value="blue"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:simpleType>
</xsd:element>
```

The enumerated values are stored as items in a pull-down list.

## Other constraining facets

Whenever possible, LiveCycle Designer maps facets that constrain bounds (minInclusive, minExclusive, maxInclusive, and maxExclusive), length (length, minLength, and maxLength), and precision (totalDigits and fractionDigits) to properties on generated fields.

Facets mapped to properties are as follows:

- The maxLength facet sets the maxChars property on form objects.
- The fractionDigits facet sets the fracDigits property on the XFA decimal value; for example:

```
<field>
  <value>
    <decimal fracDigits="3"/>
  </value>
</field>
```

Otherwise, LiveCycle Designer uses these constraining facets to generate validation scripts. The section on validation on [page 10](#) contains more information on this topic.

## Derivation by list or union

Derivation by list allows you to define elements whose values are expressed as a whitespace-separated list of single values. Such derivation generates TextEdit fields with pull-down lists.

Derivation by union allows for the union of two or more sets of values, not necessarily from elements of the same type. In general, types derived in this way map to TextEdit fields. However, when all the sets in a union are of the same type, they map to the object type associated with the type in those sets. For example, a union of sets of enumerated values would generate a pull-down list with all values from the union.

## Working with complex types

Complex type definitions can include element content, simple content, mixed content, and empty content. Complex elements do not necessarily contain attributes.

### Element content

An element that has only element content (and, optionally, attributes) generates a subform when element binding occurs. The subform contains a field for each of the element's declared attributes, as well as a field or subform for each child element.

### Simple content

Simple content refers to the content in which an element is declared to contain character data. Such content may or may not include attributes.

If an element does declare attributes, LiveCycle Designer maps it to a subform and gives the subform the same name as the element. This subform contains one field bound to the element, as well as additional fields for each attribute. The field bound to the element has the same name as the element, but with the string “\_data” appended. The additional fields have the same name as the attribute to which they are bound.

In the following sample XML Schema definition, the element <phone> has text content (a phone number can also include brackets and dashes to separate area codes and numbers) and an attribute named “type” describing several enumerated values:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="mySubForm">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="phone">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:string">
                <xsd:attribute name="phoneType" use="required">
                  <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
```

```

        <xsd:enumeration value="01 - Work"/>
        <xsd:enumeration value="02 - Fax"/>
        <xsd:enumeration value="03 - Cell"/>
        <xsd:enumeration value="04 - ISDN"/>
        <xsd:enumeration value="05 - Home"/>
        <xsd:enumeration value="06 - Pager"/>
        <xsd:enumeration value="07 - Home Fax"/>
        <xsd:enumeration value="08 - Other"/>
    </xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

```

This XML Schema definition generates a subform named mySubForm, which contains a subform named phone that contains a field named phoneType and a field named phone\_data. The phoneType field is bound to \$record.phone.type, the phone\_data field to \$record.phone.

The sample Schema definition yields the following form fragment. (To see the fields an XML Schema definition generates in LiveCycle Designer, choose Data View > Generate Fields.)

```

<field name="type" w="62mm" h="9mm">
  <ui>
    <choiceList>
      <border>
        <?templateDesigner StyleID aped3?>
        <edge stroke="lowered"/>
      </border>
      <margin/>
    </choiceList>
  </ui>
  <font typeface="Myriad Pro"/>
  <margin topInset="1mm" bottomInset="1mm" leftInset="1mm"
rightInset="1mm"/>
  <para vAlign="middle"/>
  <caption reserve="25mm">
    <font typeface="Myriad Pro"/>
    <para vAlign="middle"/>
    <value>
      <text xmlns="http://www.xfa.org/schema/xfa-template/2.2/"
">type</text>
    </value>
  </caption>
  <value xmlns="http://www.xfa.org/schema/xfa-template/2.2/">
    <bind xmlns="http://www.xfa.org/schema/xfa-template/2.2/"
match="dataRef" ref="$record.phone.type"/>
    <items xmlns="http://www.xfa.org/schema/xfa-template/2.2/" save="1">
      <text>01 - Work</text>
      <text>02 - Fax</text>
      <text>03 - Cell</text>
      <text>04 - ISDN</text>
      <text>05 - Home</text>
    </items>
  </value>

```

```

    <text>06 - Pager</text>
    <text>07 - Home Fax</text>
    <text>08 - Other</text>
  </items>
</field>

<field name="phone_data" x="2.440945in" w="62mm" h="9mm">
  <ui>
    <textEdit>
      <border>
        <?templateDesigner StyleID aped3?>
          <edge stroke="lowered"/>
        </border>
      </textEdit>
    </ui>
    <font typeface="Myriad Pro"/>
    <margin topInset="1mm" bottomInset="1mm" leftInset="1mm"
rightInset="1mm"/>
    <para vAlign="middle"/>
    <caption reserve="25mm">
      <font typeface="Myriad Pro"/>
      <para vAlign="middle"/>
      <value>
        <text xmlns="http://www.xfa.org/schema/xfa-template/2.2/">phone
data</text>
      </value>
    </caption>
    <value xmlns="http://www.xfa.org/schema/xfa-template/2.2/">
      <bind xmlns="http://www.xfa.org/schema/xfa-template/2.2/"
match="dataRef" ref="$record.phone"/>
    </value>
  </field>

```

If you choose 01 - Work as the phone type in this example, an XML instance of the element would look like this:

```

<?xml version="1.0" encoding="UTF-8"?>
<RootElement>
  <phone type="01 - Work">123-456-7890</phone>
</RootElement>

```

### Mixed content

Mixed content allows both character data and subelements. Typically, mixed content looks something like this:

```

<p>the rain in <i>spain</i> falls mainly in <b>the</b> plain</p>

```

Mixed content typically occurs when end users enter raw markup in text fields. As a result, mixed content is unlikely to occur in form data. LiveCycle Designer assumes that mixed content indicates emphasis in character data and therefore treats mixed elements as having no subelements. The effect is somewhat unstructured from a form point of view but is useful in a markup language such as XHTML.

Note, however, that if an element formally defines content from the XHTML Schema definition, LiveCycle Designer treats the content as rich text. This situation is described in the section on rich text fields on [page 8](#).

## Empty content

LiveCycle Designer treats complex types with empty content the same way it treats those with simple content, creating a field and binding it to the element. However, the field has no allowed content.

## Occurrence

LiveCycle Designer generates repetition properties for XML Schema elements with occurrence attributes (minOccurs and maxOccurs) that generate subforms.

## Content models

If the content model for sibling subelements is other than a sequence, LiveCycle Designer attempts to model choice and all groups using XFA subformSets.

As a general rule, each content model maps to an equivalent subformSet. A sequence content model often makes this mapping redundant because a subform with nested subforms uses them in form order by default. In this case, LiveCycle Designer drops the subformSet to reduce file size and overhead. If a choice model includes a sequence model as one choice, LiveCycle Designer generates a subformSet for that sequence.

Typically, data order and a subformSet correspond to set and choice models. The dataDescription represents these with a dd:group element with an attribute indicating the type of group.

## Understanding the effects of element specifications

This section describes how LiveCycle Designer handles element specifications in XML Schema definitions.

### Rich text fields

The default mappings for the TextEdit field (see [Table 1](#)) generate fields with plain text content.

LiveCycle Designer maps elements defined as having content from the XHTML Schema definition to rich TextEdit fields rather than plain TextEdit fields. In addition, the <field> object type created in the form has the value:

```
<value><exData contentType="text/html"/></value>.
```

There are three situations in which elements are identified as having rich text content:

- The XML Schema definition imports the XHTML Schema and declares an element containing a single child that is the XHTML <body> element. An example of such an element is:

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:xhtml="http://www.w3.org/1999/xhtml">
<xsd:import namespace="http://www.w3.org/1999/xhtml"/>
  <xsd:element name="RichTextField">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="xhtml:body"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

- A Schema definition declares a string-type element with a default or fixed value that begins with this content:

```
<body xmlns="http://www.w3.org/1999/xhtml" ...
```

In this case, LiveCycle Designer creates a rich text field and sets its value to that of the default or fixed value.



- An element declaration includes an attribute of `xfa:contentType` with a fixed value of `text/html`, for example:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xfa="http://www.adobe.com/2003/xfa"><xsd:import
  namespace="http://www.xfa.org/schema/xfa-template/2.2/" />
<xsd:element name="RichTextField">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute ref="xfa:contentType" fixed="text/html" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

In this example, LiveCycle Designer does not use the `xfa:contentType` attribute to generate an XFA field; instead, it interprets the attribute as a directive that the field generated should be a rich text field. Note that the actual namespace identifier for XFA is yet to be determined.

## Images

LiveCycle Designer maps elements declared as images via XFA attributes to Image fields in the form. The XFA attributes are:

```
xfa:contentType xfa:href xfa:transferEncoding
```

This sample code makes such a declaration:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xfa="http://www.adobe.com/2003/xfa"><xsd:import
  namespace="http://www.xfa.org/schema/xfa-template/2.2/" />
<xsd:element name="ImageField">
  <xsd:complexType><xsd:simpleContent>
    <xsd:extension base="xsd:hexBinary">
      <xsd:attribute ref="xfa:contentType" fixed="image/jpeg" />
      <xsd:attribute ref="xfa:transferEncoding"
        fixed="base64" />
    </xsd:extension><xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

In this case, LiveCycle Designer does not use the XFA attributes to generate fields in the form; instead, it uses the attributes to recognize elements as corresponding to Image fields and set any default or fixed values for the attributes on the generated XFA `<image>` element.

## Default and fixed values

Attribute and element declarations that have a simple type or complex type with simple content may specify a fixed or default value for the element or attribute.

An element or attribute declaration that specifies a fixed or default value generates a field in which the initial value is set to the fixed or default value specified in the declaration.

## Annotation information

LiveCycle Designer maps annotation information declared as the child of an `<xsd:element>` or `<xsd:attribute>` in the XML Schema definition to an XFA field that is generated from that declaration as XFA `<desc>` content.

The mapping of annotation information is optional and can be controlled by the Data View's flyout menu. See the LiveCycle Designer documentation installed with the product for more information.

## Namespaces

If the XML Schema definition includes a `targetNamespace`, LiveCycle Designer preserves this information and uses it to qualify data elements. The namespace and namespace prefix information does not appear in the Data View tree or in the fields or subforms generated from the data connection. However, the namespace information is preserved in the `dataDescription` generated from the XML Schema definition. XML written from the form uses the namespaces according to the declarations in the XML Schema definition.

## Schema composition

LiveCycle Designer supports the use of `<xs:include>` and `<xs:import>` in an XML Schema definition to generate a composite schema provided the include or import elements specify a `schemaLocation` attribute whose value is an absolute or relative file path on the local file system.

In the case of `<xs:import>` support, LiveCycle Designer correctly preserves target namespace information for all imported components in the `dataDescription` generated from the composite schema.

## Substitution groups

LiveCycle Designer supports substitution groups, which are commonly used with the `<xs:import>` feature. Where substitution groups are used in XML Schema definitions, LiveCycle Designer maps any head element in a substitution group to a choice group in the Data View menu and corresponding `dataDescription`. Members of the choice group include the head element and all possible substitute elements. If the head element occurs in a Schema-defined choice group, substitute elements appear as additional choice options in that group, and no additional choice groups appear.

## Using validation scripts

Where an XML Schema definition provides information about permitted values of an element or attribute and therefore of the corresponding field in a form, the field includes a JavaScript validation script that determines whether the Schema declaration allows that value. The data type of the element or attribute can provide this information about values, as can restriction facets applied to the base type. Consider the following example:

```
<xsd:element name="someNumber">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="4"/>
      <xsd:maxInclusive value="9"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The mapping creates a numeric field named `someNumber` and a script that validates its content as an integer between four and nine inclusive.

Many of the restriction facets in an XML Schema definition provide information you can use to generate a validation script. However, you can do this only if the information in the facet does not map directly to an XFA property—for example, as the `maxLength` facet for a string type maps to `<text maxChars="">`.

Note that generating validation scripts is optional. Use Field Generation Options in the Data View's flyout menu to set your validation options (Figure 2).

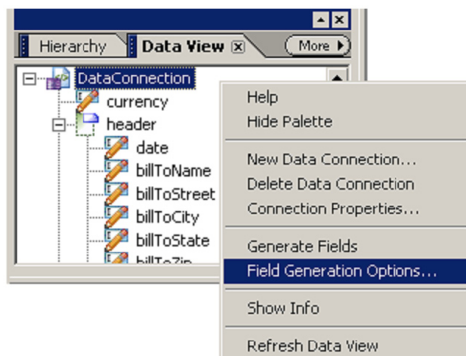


Figure 2: Controlling validation options

### Validating numeric types

LiveCycle Designer maps all numeric types in an XML Schema definition to a single numeric type in XFA and sets the Data Format type to Integer, Decimal, or Float. Decimal types may have the `fracDigits` property.

You can maintain restrictions imposed by XSDL numeric types by generating validation scripts to ensure that a numeric value is in the range allowed by the type. For example, an unsigned byte type generates a numeric field plus a script to validate that the value is an unsigned byte value, such as the following:

```
this.rawValue >= 0 && this.rawValue <= 255;
```

Appropriate range testing scripts will be generated for integer, positive integer, negative integer, non-positive integer, non-negative integer, long, int, short, byte, unsigned long, unsigned int, unsigned short, and unsigned byte datatypes.

### Validating inclusion and exclusion

When a numeric type specifies min or max inclusion or exclusion values or both, LiveCycle Designer can generate a script to test that the value entered is within the range defined by these restriction facets. If a field already has a script to validate numeric ranges (see the section on validating numeric types), a script to validate inclusion or exclusion will test the more restrictive range.

### Validating length, minLength, and maxLength

An XML Schema definition string type may specify a length, `minLength`, or `maxLength` restriction facet. If the XML Schema definition node maps to a `TextEdit` field, LiveCycle Designer uses the values specified in the length and `minLength` restriction facets to generate a script to validate that the `TextEdit` length is the value specified by length or is at least the value specified by `minLength`.

Note that any `maxLength` facet value maps as `maxChars` for the `TextEdit` field.

Note also that when a string type maps to a pull-down list type in XFA because it has enumerated values, the generated validation script uses length, `minLength`, or `maxLength` values to check that the length of a user-selected value is valid according to the length, `minLength`, and `maxLength` facets.

### Validating totalDigits

The `totalDigits` facet specifies the maximum number of digits that can be used for a decimal or any of the following datatypes: decimal, integer, positive integer, negative integer, non-positive integer, non-negative integer, long, int, short, byte, unsigned long, unsigned int, unsigned short, and unsigned byte datatypes.

If an XML Schema definition specifies `totalDigits` for an element or attribute that maps to a numeric field, LiveCycle Designer can generate a validation script to test that the total number of digits is as defined by the restriction facet.

## Getting started with LiveCycle Designer

If you have an XML Schema that describes the data you want your form to generate, you are ready to use it to define fields in your form file. The following steps describe the general process for binding XML Schema elements to a form file:

- To create a data connection in LiveCycle Designer, choose **File > New Data Connection** from the menu bar. In the dialog box presented by the LiveCycle Designer wizard, select **XML Schema**. Alternatively, you can select a sample XML data connection if you do not have an XML Schema definition. A series of dialog boxes will walk you through the creation of the data connection.
- Bind the data in the data connection to the fields in the form design. To create the fields automatically, drag all or part of the data connection onto your form. If you have an existing form with fields on it, you can drag and drop individual elements in the **Data View** onto the appropriate field. Field names do not change, and the binding property is the only property that changes.

- View and edit the binding properties on the Object palette Binding tab. Binding is described by a Scripting Object Model (SOM) expression such as \$record.name. For more information on the SOM, refer to the Scripting Object Model Expression Specification at [http://partners.adobe.com/public/developer/en/xml/som\\_2.0.pdf](http://partners.adobe.com/public/developer/en/xml/som_2.0.pdf).

For more information on developing solutions with LiveCycle Designer, visit [http://partners.adobe.com/public/developer/livecycle/topic\\_designer.html](http://partners.adobe.com/public/developer/livecycle/topic_designer.html) and follow the link to the LiveCycle Designer forum.

## Conclusion

Form designers use LiveCycle Designer to create or adapt electronic form elements based on XML Schema definition elements. XSDL makes possible the definition of data types and content constraints that LiveCycle Designer turns into form fields and subforms. End users experience the interactivity of tightly coded forms with pull-down selection lists, radio buttons, checkboxes, and display patterns, as well as the validation of their entries. Form designers can control LiveCycle Designer's form generation process and define the presentation features they require.

Using XML Schema definitions in LiveCycle Designer simplifies electronic form design and helps ensure the consistency of electronic form designs and the consequent integrity of the data in those forms. The process moves responsibility for data definition from form designers to schema designers, centralizing enterprise-level design decisions where they are most effective.

Adobe helps people create, manage, and deliver the highest quality digital content in the world.  
**Better by Adobe.™**

**Adobe Systems Incorporated**  
345 Park Avenue, San Jose, CA 95110-2704 USA  
[www.adobe.com](http://www.adobe.com)

Adobe, the Adobe logo, Acrobat, and Adobe LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Mac and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. PowerPC is a registered trademark of IBM Corporation in the United States. Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners.

© 2005 Adobe Systems Incorporated. All rights reserved.  
Printed in the USA.

08/05

