1

About InfoPath

In the spring and summer of 2002, Microsoft started showing pre-alpha versions of what was then called XDocs to selected corporate customers. Something interesting was on the way. XDocs was far from finished, and it wasn't certain how what is now InfoPath would be positioned or how it would fit into the rest of the Office product line. Some limited XML features were already present in the existing Office applications, and it was reasonable to expect enhancements in that area in the next major release. It was also clear, even then, that InfoPath was going to be something of a departure.

If InfoPath joined the Office application suite, it would be the only product without a considerable pre-XML legacy, and there was an opportunity to make a fresh start in introducing XML compatibility to part of the product line. It also seemed, as is still evident, that InfoPath would initially be much more dependent on developer skills than anything else in Microsoft Office 2003.

As time passed we learned that support for XML in the 2003 versions of Access, Excel, and Word would be expanded considerably from what was initially available in Office XP. The missing piece was the fit for InfoPath. In retrospect it seems obvious. InfoPath would be a new information-gathering program using XML as its native file format.

But why does Office need yet another XML processor? With the new features in Word, we can create custom XML documents. And Access 2003 and Excel 2003 will now do a good job of capturing regular data structures in any schema we choose. The answer lies in *forms*, possibly the last arena in office systems that is pretty much untouched by XML technology.

Initially, because of its inheritance from SGML, XML was seen as an enhancement that would benefit online document-oriented applications. Then XML was adopted, some think hijacked, by developers who wanted it as an interoperable format to oil the wheels of e-commerce, and there's no doubt that data-oriented XML has recently been the primary driver of Internet standards. Forms sit somewhere between the two poles of document and data orientation. Whereas documents can have extremely complex information structures, including features such as repeating elements and recursion, regular structures like database tables and spreadsheets are simple and straightforward. Office forms can combine the two features. They are usually quite short but often take a semistructured form, combining simple field lists with optional sections and repeating elements—for example, the dates and details in an expense claim.

Chapter 1

XML Forms

Consider first the very general example of the expense or travel claim. Suppose Human Resources has given you an electronic copy of an Excel sheet that will work everything out for you. You fill in the blanks and print it out. Then you put it on your manager's desk so she can sign it. Eventually, it gets to Payroll, where someone else enters some or all of the data again.

Now you perform the same task, this time using InfoPath. Instead of a spreadsheet, you download a form template to complete. It also does the necessary calculations. You e-mail it to your manager, who approves it with a digital signature and routes the form to anyone else who needs to sign. The XML data is harvested by the payroll system, and the repayment gets added to your pay slip in time for the big weekend you have planned. This by itself is probably a sufficient motive for any developer to learn InfoPath and implement a new staff expenses system.

From a less selfish perspective, think of the thoroughly forms-intensive business processes where data is still bound up in paper-based systems. If you have ever worked for an insurance company, a financial services firm, a hospital, or a government department, you'll see the huge potential in unlocking the data carried in office forms.

But without a doubt the most attractive feature of InfoPath is that you can hide the complexities of XML from end users. Even if you understand XML, it can get in the way. A while ago, one of us explored the idea of introducing XML capture for a large group of developers working on a complex API. It would have made the creation of an HTML reference easy, but while everyone saw the validity of the business case, they rebelled at the thought of using a traditional XML editor. If InfoPath had been available then, no doubt they would have been more supportive.

Microsoft isn't the first, let alone the only, vendor to spot the opportunity for forms tools, and there will be plenty of competition for this very large market segment. Microsoft may have an advantage, however, because of its dominant position in the office market.

To put InfoPath in context, we suggest you take a few moments to look at the alternatives there are to the approach that Microsoft has taken. Several observers and commentators have compared InfoPath to XForms, a recent W3C Draft Recommendation intended to be integrated into other markup languages, such as XHTML or SVG. See, for example, Michael Dubinko's *XForms and Microsoft InfoPath* at `www.xml.com/pub/a/2003/10/29/infopath.html`.

Perhaps the comparison is made because there is an implied expectation that forms processing should mainly follow the Web processing model. You may or may not agree that the Web is the natural home for forms, but in any case a direct comparison just isn't productive. As Dubinko points out, that's because InfoPath is an application, whereas XForms, together with a number of other interface markup languages, is an XML vocabulary.

It may be more helpful to look at points of similarity and difference adopted by developers of XML forms applications, including XForms.

Common Features in XML Forms

When it comes down to it, XML forms processors have more in common than you might think. Essentially, they are there to convert user input into new or modified XML data, which can then be routed through a series of business process, possibly on multiple platforms in different organizations.

A central design concept is a "package" of files with distinct functions: a template document with a structure definition from a fixed, industry-standard or custom XML schema; an XML file to contain default data; and form data in an XML file that can be routed to points in a workflow. At each point, the data is loaded into a form, which provides a view into editing all or parts of the form. This process can be repeated as many times as necessary, with any number of participants.

Some approaches, like XForms, use fixed element names for controls and encourage implementers to define the purpose of the data-gathering controls. This makes it easier to generate the related structures automatically, for example, creating different interface objects for PDA and desktop browsers. Others are more focused on providing a rich user interface (UI). However, most have a wide range of display properties that include showing or hiding parts of forms and repeating sections where elements can be added or removed by users.

Points of Difference

Probably the first distinction to note is between Web-based XML forms and rich client systems. Web-based applications have their attractions, and on office intranets they have become a common way to collect some kinds of information from employees. They are easy to deploy and inexpensive to support. But thus far they have not been good candidates for workflow processing. That will soon change as XForms-based tools appear.

Rich client applications are relatively expensive to deploy and maintain, but they are often more robust and can be more readily integrated with other desktop client systems. They can be operated when users are disconnected from the network, and there is some evidence that users prefer them to Web-based tools when they have a choice.

Another distinction is between declarative and scripting approaches. A goal of the XForms specification was to limit the need for scripting; it therefore makes use of XPath-based calculation and validation, and includes XML `action` elements that specify responses to events like setting focus or changing a data value. In contrast, InfoPath, although it makes some use of declarative programming, including XPath expressions, encourages the use of script more often.

InfoPath Features in Outline

Later in the book we'll discuss InfoPath features in greater detail, but for now, here's a summary of some of the key XML technologies and development approaches.

XML from the Ground Up

InfoPath applies a range of XML technologies recommended by W3C that we noted in the Introduction. This is a first for Microsoft, and thus a first for you as an Office developer. Additionally, InfoPath makes use of XML processing instructions and namespaces. There are also methods for accessing the XML document using the InfoPath Object Model (OM).

The following table outlines the use of some XML standards applied in InfoPath.

Name	Description
XML	XML is the format that underlies an InfoPath form.
XSLT	XSLT is a specification for transforming XML files. It is the format of the View files that are produced when a form is designed. The transform creates an XHTML document that is displayed in the user interface.
XML Schema	XML schemas provide the underlying structure of the XML form and are the primary means of data validation. XML Schema is used to define the structure of the form definition. See Appendix A.
XHTML	XHTML is the XML-conformant version of HTML. In InfoPath it is used to display formatted text in rich text controls.
XPath	XPath expressions are used to bind controls to forms. XPath is also used in data validation, conditional formatting, and expression controls.
DOM	The Document Object Model is primarily used in scripts to access the contents of the form document, but it can be used with any XML document in the InfoPath environment.
XML Signature	XML signatures are used to digitally sign InfoPath forms created by. Forms can contain multiple signatures.

Some Constraints

Although InfoPath supports a wide range of XML features, there are some limitations or other constraints in this release that you should note. InfoPath 2003 does not support the following:

- XML Schema constructs `xs:any`, `xs:anyAttribute`
- `abstract` and `substitutionGroup` attributes on elements and types
- XSL Formatting Objects (XSL-FO) for the presentation of XML data
- Import or inclusion of arbitrary XSL files
- XML-Data Reduced (XDR) or Document Type Definition (DTD) for defining schemas
- Digital signing of parts of a form
- XML processor versions earlier than Microsoft XML Core Services (MSXML) 5.0

About Form Development

We've already noted a bit of a bias on the part of Microsoft toward code to extend the functionality of forms. This shouldn't be surprising, because Microsoft and therefore many developers of Microsoft applications have come to XML quite recently and have come from a code-based programming background. The structures found in XSLT, for example, can often seem obscure and foreign. As XML becomes a more prominent component in Office applications, we may come to see that declarative programming approaches are increasingly common.

When you modify InfoPath forms, by setting values in design mode or by editing values in the form files with a text editor, you are customizing the form *declaratively*. When you alter a form *programmatically*, you are writing code using JScript or VBScript following the InfoPath Object Model.

As we've mentioned already, there are often two or three ways to achieve the results that you want.

Declarative Development

Declarative development involves modifying one or more XML files, including:

- XML Schema that defines the structure of the form
- XSLT files that define the views on a form
- XML form definition file or *manifest* that specifies the overall structure of a form

Why might you want to use the declarative approach? Well, one reason might be that you prefer it to programmatic development in certain cases, but there are times when InfoPath leaves you few options. Here are just some occasions when declarative programming is either recommended or necessary:

- Custom form merging
- New menus and toolbars
- Schema and other upgrade modifications
- Custom transform templates
- Adding processing instructions to XML data files
- Exporting form data to custom Excel schemas
- Creating custom task panes and their associated files

Programmatic Development

You can customize a form programmatically by writing scripting code to perform a variety of functions. The main components that involve programmatic interaction are listed in the following table.

Component	Description
Object model	Type library composed of collections, objects, properties, methods, and events that give you programmatic control of the environment.
Data validation	XML schemas, expressions, and scripting code used to validate and constrain the data that users are allowed to enter in a form.
Event handling	Event handlers that respond to form loading, changes to content, view switching, and implement custom form submission.
User interface	Customizable interface components including menus and toolbars with related buttons, command bars, and a task pane.
Editing controls	Controls that include collections, optional items, text lists, and fields.

Table continued on following page

Component	Description
Error handling	Event handlers, OM calls, and form definition file entries used to handle errors.
Security	Security levels that restrict access to the OM and system resources.
Data submission	Predefined functions that can be used to implement custom posting and submission.
Business logic	Scripts to implement editing behavior, data validation, event handlers, and control of data flow. The logic can also access external COM components.
Form integration	Integration with other Office applications and SQL Server, SharePoint, or XML Web services.

Microsoft Script Editor

InfoPath includes the Microsoft Script Editor IDE (Integrated Development Environment) creating and debugging code, together with programming languages that you can use to extend your applications. However, while other Office applications use Visual Basic for Applications (VBA) as their primary programming language, InfoPath uses two scripting languages—JScript and VBScript.

JScript is an interpreted, object-based language that is the Microsoft implementation of the ECMA 262 language specification. VBScript is a subset of the Microsoft Visual Basic. However, while you have a choice of languages, you cannot mix the two languages in a single form.

You can set the default language for a form in the design mode interface. When you open Microsoft Script Editor (MSE) from InfoPath in design mode, the MSE code editor appears and the form's default scripting file opens in the code editing window.

In debug mode, you can use all of the debugging features that MSE provides, including using breakpoints, stepping through program statements, and viewing any of the debugging windows.

Introducing the Design User Interface

InfoPath has separate modes for the two distinct tasks of filling out and designing forms. Design mode is the environment in which you create or modify a form template.

Before getting into the details of form architecture in the next chapter, we'd like you to take a quick look at the InfoPath user interface, with an emphasis on the design mode features. So if you haven't already done so, why not start InfoPath and take a short tour with us?

Start by opening the form template called Absence Request that comes with InfoPath. To open the form, click the `More Forms` button on the `Fill Out a Form` task pane, and choose the form on the `Sample Forms` tab of the dialog box. The usual path of the form template files is C:\Program Files\Microsoft Office\OFFICE11\1033\INFFORMS\1033\.

The workspace is divided into two main areas: the form area and the task pane area. The form area appears on the left side of the screen, and the task pane area by default appears on the right, regardless of whether you are designing a form or filling it out. Figure 1-1 shows the Absence Request form with the Help task pane.

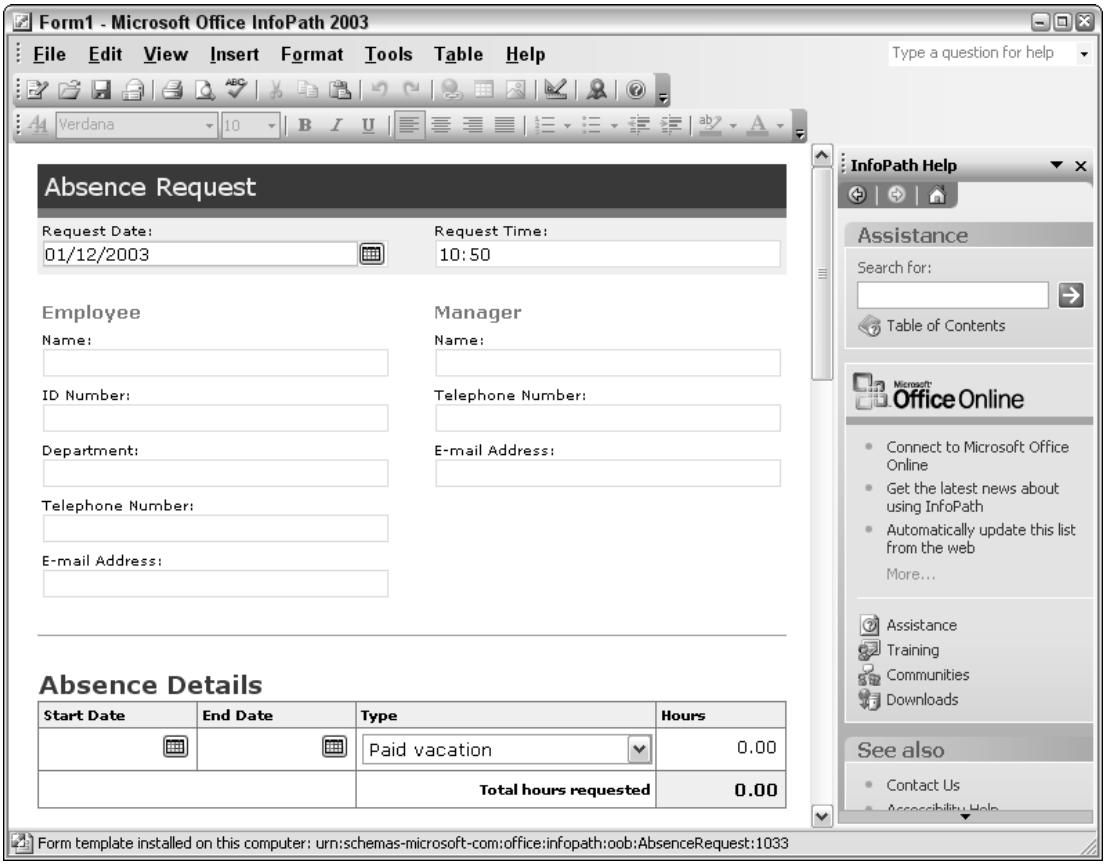


Figure 1-1: The InfoPath workspace.

The Form Area

The form area is where you enter your form data. When you fill out a form, the form template's location is displayed at the bottom left of the form area.

In design mode, a form appears in the same position, but you can access the form controls, set their properties, and define other form options. To switch between fill-out and design modes, you can either choose an option from the drop-down menu at the top of the task pane, click a toolbar button, or choose `File➪Fill Out a Form` or `Design a Form`. Figure 1-2 illustrates the form area in design mode.

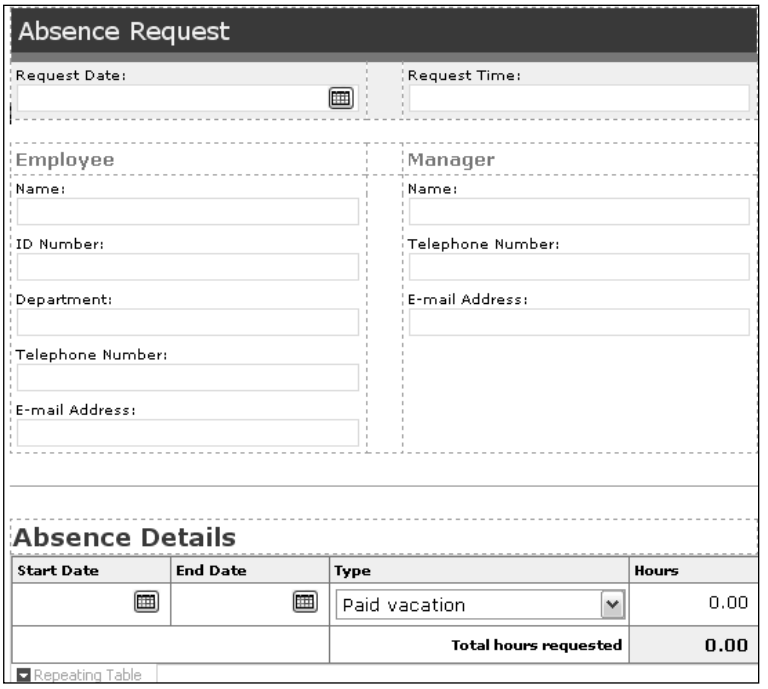


Figure 1-2: The form area in design mode.

The Task Pane Area

You can locate the task pane anywhere you like; it can be moved, resized, docked or floated, opened or closed, just like other Office applications. Depending on the context, task panes can contain commands for switching views, formatting options, and inserting form controls. They can also contain, help text, hyperlinks, or clip art.

When you design a form, you can create custom task panes that will be available to your users when they fill out your form. Custom task panes consist of an HTML file and can also contain form-specific content, such as command buttons and data catalogs.

When you design a form, your options include choosing between creating a new form or modifying a sample. New forms can be built from scratch, starting with a blank form and adding controls. If you create a new form from a data source, you have the option to base the form on an XML schema, a database, or a Web service. See Figure 1-3.

Design Tasks

The design mode task pane, shown in Figure 1-4, provides several options, three of which you'll review here:

- Layout
- Controls
- Data Source

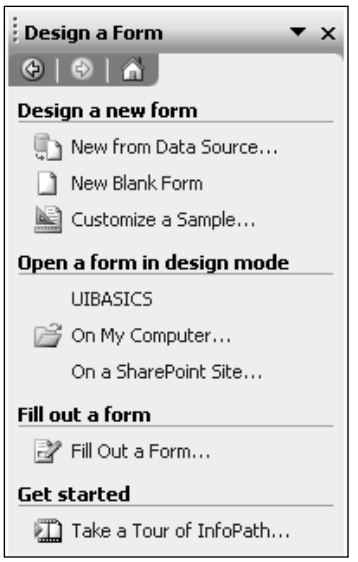


Figure 1-3: The task pane in design mode.

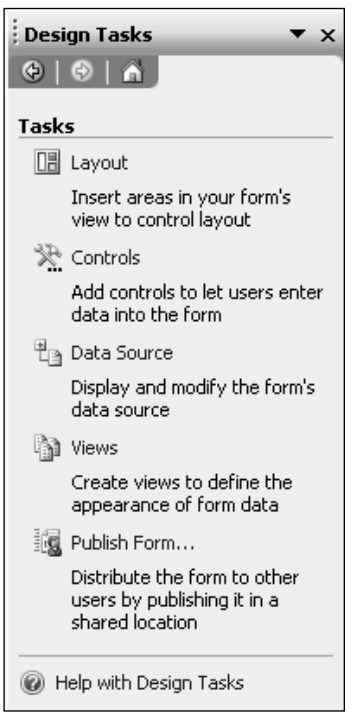


Figure 1-4: The task pane showing design tasks.

Layout

InfoPath provides for several tabular layout options, with four useful commonly used layouts that can be modified and one for a custom table. When you insert a table layout, InfoPath draws a dotted line around the table cells.

There are commands or task pane buttons to merge and split cells, add and remove columns and rows, and so on. Figure 1-5 shows part of the Layout task pane.

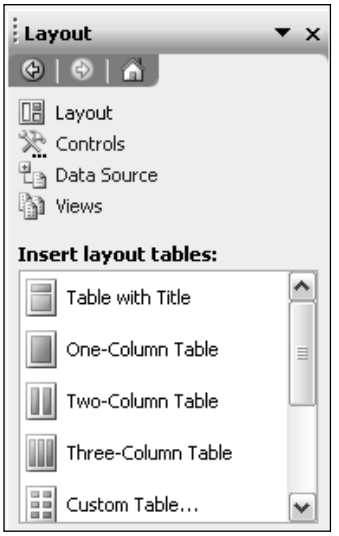


Figure 1-5: The Layout task pane showing table options.

Controls

When you click the `Controls` button, you can choose from a rich set of form controls (see Figure 1-6). Any control can be dragged onto a form. If you are using a blank form, InfoPath gives the control a default name and creates a related form field to which the control is bound. It also creates an element with that name in the form schema.

You'll learn more about the available controls and how to use them as you read through the book, especially in Chapter 8.

Data Source

When you click the `Data Source` button, InfoPath displays a hierarchical list of the fields to which the form controls are bound. Fields store the data that you enter into controls. Figure 1-7 shows part of the list from the Absence Request form.

You can set or view the properties of a data source field by right-clicking the field in the data source list and choosing `Properties` from the context menu. The Properties dialog box is displayed automatically if you have created a form from a schema, and you drag a new control onto the form.

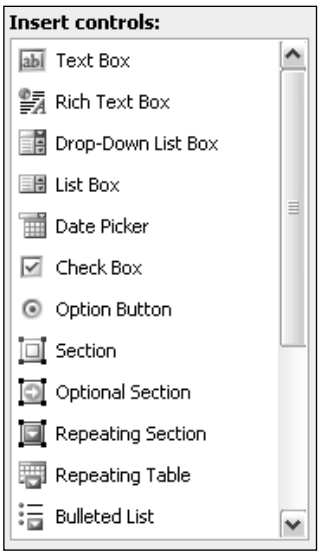


Figure 1-6: The controls list.

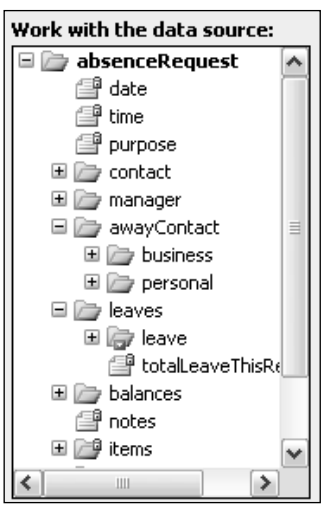


Figure 1-7: The form data source list.

Figure 1-8 illustrates the properties of the XHTML `notes` element in the Absence Request form. Note that most controls in the dialog box are disabled. This is because the XML schema for the form predetermines the properties, such as name and data type, and you can't change them unless you modify the schema itself.

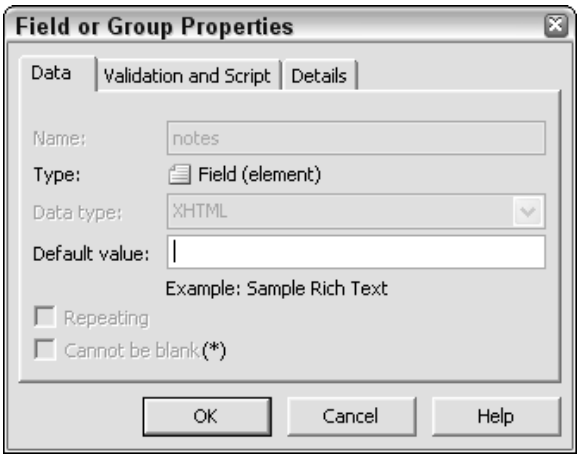


Figure 1-8: The form data source list.

Design Mode Icons

InfoPath makes extensive use of icons to give you feedback on objects in the design interface. It is worth spending a few minutes reviewing them, because it will help you understand the relationships between the form objects and the underlying XML document.

Data Source Icons

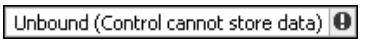
A field represents an XML element or an element attribute. If a field represents an element, it can contain attributes fields. The data source list also shows groups, which can contain fields and other groups. Groups typically contain controls like repeating tables and sections.

The table shows the three basic icons and their meanings, together with the decorations that are added for repeating, locked, and required properties.

Icon	Name	Meaning
	Group	An element in the data source that can contain fields and other groups.
	Element field	Stores the data entered into controls. This field can contain attribute fields.
	Attribute field	Stores the data entered into controls. Other fields cannot be added to it.
	Repeating field	May occur more than once. Lists, repeating sections, and repeating tables can be bound to repeating fields.
	Locked field	Cannot be moved or deleted.
	Required field	Cannot be left blank.

Binding Icons

In design mode you can tell if a form control is correctly bound to an element or attribute in the data source. Icons are displayed in the top right corner of the control if a control is not bound to a field or there is a duplicate binding. If the binding is correct, the icon shows only when you mouse over the control. The following table shows the icons and their meanings.

Icon	Meaning
	The binding is correct.
	The control may not function as expected because more than one control is bound.
	The control will not function correctly because it is unbound.

Summary

In this introduction to InfoPath, you looked at how InfoPath fits into the new developments in XML forms processing and saw how desktop applications compare and contrast with those designed around declarative approaches like XForms. You also reviewed the use of XML standards in InfoPath and some of the constraints that exist, and followed an outline of the programming environment. As a prelude to the overview of InfoPath architecture in Chapter 2, you dipped into some of the interesting features of the design mode user interface.

Now it's time to start working.