Robert L. Bogue
MS MVP, MCSE, MCSA:Security
317-844-5310
Rob.Bogue@ThorProjects.com

Coding Options for Building InfoPath Forms

Decisions, decisions. How do you add code to your InfoPath forms or decide whether putting code in them is even necessary? Learn the basics to decide what's right for your forms.

When you want to build a simple form—something with a set of fields and perhaps a table of repeating fields—it's easy to pick up InfoPath and bang out a form in a matter of minutes, or at least within an hour. However, integrating code with your InfoPath forms can make form building a trickier proposition. Sure, VB.NET and C# are supported languages for extending InfoPath forms, but how does that support work and how are you going to be able to deploy it?

This discussion will allow you to investigate options for developing code when using InfoPath, consider the limitations, and discover ways to make intelligent, code-enabled InfoPath forms.

A Cornucopia of Coding Options

At first glance your options for developing code with InfoPath look simple. You can use either script or .NET code, namely C# or VB.NET, but in addition to language choices there are two different approaches that you can take for .NET development. You can use either Visual Studio Tools for Applications (VSTA) or Visual Studio Tools for Office (VSTO). VSTA ships with InfoPath and doesn't require a Visual Studio license, but VSTO does. As you explore the coding options for InfoPath, consider these .NET development environments first.

The decision between VSTA and VSTO is largely akin to making a decision about what brand of car to buy. Both will get you to your destination—a smart form—but they have different characteristics in terms of their focus and approach. VSTA is a sidecar to InfoPath; it takes the InfoPath interface and binds to it to manage the code's development.

You can use VSTA to design the form in InfoPath, and then you can write the code using the separate VSTA tool. This approach seems like a major inconvenience at first, but after a short time it becomes fairly natural. The nice part of VSTA is that if you're working on just the form's layout you don't have to open VSTA. You start InfoPath and away you go.

VSTO on the other hand integrates the InfoPath designer as a part of the Visual Studio experience; therefore, you can transition between designing the form and writing code for the form without leaving the tool. If you're approaching the form as a developer who happens to use InfoPath, then using VSTO may be the better solution. It supports source code control, even if it can be quirky.

The problem with VSTO is primarily that it's InfoPath designer tool doesn't reproduce InfoPath forms with 100 percent fidelity. If you can live with the quirks and have the extra time to learn more about it, VSTO is probably a better tool for long-term use.

A Tale of Two APIs

Note that the foregoing discussion doesn't address the API model. InfoPath 2003 used a COM model that communicated through MSXML; however, InfoPath 2007 uses .NET instead. The two APIs are incompatible. You can get tips on how to convert from the 2003 model to the 2007 model by reviewing "Converting InfoPath 2003 Managed Code to the New InfoPath 2007 Object Model" (MSDN, June 2007). Just be aware that conversion is a manual process.

Perhaps the greatest changes in the object model are to the names of the *root* objects, such as things like the *thisXDocument* object disappearing and the change from an XML DOM-based reference model to an XML Navigatorbased reference model. There are other changes, as well, such as how events are wired up. However, these events pale in comparison to the problems with learning a new set of root objects—and a completely new way of working with XML.

In regard to the change in root objects, the transitions aren't that hard after you get used to them, but if you've come from an InfoPath 2003 environment you may find yourself trying to use objects that are no longer there. When you go to reuse your previous code, or code you find on the Internet, it may be temporarily frustrating to learn how to convert the code into the 2007 model.

The transition to XML Navigator objects means you're no longer working with XMLNode objects. In truth, XML Navigator objects aren't really all that difficult. They're simply a pointer into the XML document, which can take a bit of getting used to because it's a new way of working with XML. For example, instead of extracting nodes using XPath statements, you move a pointer—a bookmark—in the document with a series of *move* statements or XPath queries.

The real reason why you should care about the differences between the InfoPath 2003 and InfoPath 2007 object models is because despite the fact that the InfoPath 2007 client will support both object models, InfoPath Forms Services supports only the 2007 object model.

When you're confronted with an organization that deploys InfoPath 2003 and Microsoft Office SharePoint Server Enterprise Edition (MOSS-Enterprise), you encounter a fork in the road. You must decide to either use the 2003 model and use the InfoPath 2003 client, or use the 2007 object model and require that the form be filled out through Forms Services. This approach is complicated by the fact that InfoPath 2003 doesn't work with MOSS content types. Therefore, if you want to create a content type for InfoPath to use, you have to essentially use InfoPath Forms Services or the InfoPath 2007 client. Obviously, this problem will fade over time as more organizations replace InfoPath 2003 by deploying InfoPath 2007. The real trick is to make the decisions up front about what limitations that you're willing to live with and which ones you aren't.

InfoPath supports script too, and it's quite handy when working with task panes; however, there is a set of limitations around script that render it an unpopular first choice among many developers. For one thing, you can have only one type of code per project; that means that if you want .NET code you can't use script. You also cannot mix JavaScript and VBScript in a script project—even though both are supported individually. Script is also not supported in Forms Services, which means it's off limits if you plan to publish the form for use with Forms Services—and one could easily question the exclusion of script in Forms Services support.

A Matter of Trust

In today's world it's important, if not critical, to trust the code you're running. With viruses, worms, and Trojans coming out of the woodwork, you can't afford to run code today that you didn't write, didn't verify, or for which you don't at least trust the source.

The publisher sets the security level of an InfoPath form, and it is confirmed by the consumer of the form template. To set the options from the category list, the designer selects Tools --> Form Options, and then on the Security and Trust pane unchecks the "Automatically determine security level (recommended)" option and manually selects the level that is appropriate for the form.

InfoPath Security

InfoPath has built-in mechanisms to limit the amount of trust that you can apply to a form. There are three security levels: *restricted*, *domain*, and *full*. These security levels control what the form can and cannot do.

In the *restricted* security level the form can't access data outside of the form itself. In other words, all the integration with the environment (with the exception of email submission) is disabled. This level of security is not supported for Forms Server templates; they're promoted automatically to the domain level of security. When you add an email data connection for submission, InfoPath will automatically set the form to a *restricted* security level.

Domain-level security allows access to the data sources in the same domain, on the local computer, and in the local intranet zone as specified in Microsoft Internet Explorer. It's limited to utilizing built-in integration functions and some lower-level code only. Sure, forms with *domain*-level security can run code, but only certain methods. The article "About the Security Model for Managed Code Form Templates" (from "InfoPath Developer Reference for Managed Code" MSDN, 2007) describes in general which objects and methods are available for domain-level trust forms.

The highest level of trust, *full* trust, is necessary when the code in the form needs unrestricted access to the API. If you think about it, after you're running some code, it's possible to do nearly everything to the computer, including things that a form shouldn't be able to do. However, *full* trust requires that the form is signed either with a code-signing certificate or installed on the computer. For a workstation this publishing happens through a standard Microsoft Windows Installer MSI file. On a Forms Server it's published as an administrator-approved form template.

Code signing works by indicating the verified author of the form. This technique allows the user to decide whether or not to run the form. The code-signing process relies on a trust hierarchy and a secure hash. The trust hierarchy basically means that there is a set of trusted certificate authorities; these are authorities that you believe will tell you the accurate identity of a certificate holder. Microsoft provides a set of trusted authentication providers out of the box; these are organizations that Microsoft believes you should trust to provide accurate verification of the people they issue certificates to. In addition, if you're a member of a domain and there is an enterprise certificate authority, you'll also trust the domain's enterprise certificate authority.

This trust hierarchy is coupled with a secure hash, which can be recalculated by every computer. The hash is calculated by the signer and then encrypted with its private key. The signer then attaches a copy of its certificate that includes its public key. The certificate itself includes the hierarchy of servers that issued the certificate. The recipient computer calculates the hash for the document and decrypts the hash in the document with the public key from the certificate. If the keys match, then the owner of the certificate—issued by a trusted source—is indeed the person who signed the code. This approach means that if you trust the person or organization that was issued the certificate, you should trust that the code came from them.

Code-signing certificates aren't difficult to obtain, and you can purchase them from one of the trusted root certificate authorities. Or, if you're in an organization with an active directory domain, you can issue one yourself. (You can read step-by-step instructions on how to issue a certificate in the blog post "Domain Certificate Authority Signing InfoPath 2007 Forms.")

Approved Form Templates

Administrator-approved form templates are templates that have been published by the administrator of a Forms Services computer. The administrator of the server has specifically decided these forms are appropriate for users of the Forms Services server. Because there is a trusted person—namely, the administrator specifically approving form templates published to the Forms Services server—using InfoPath form templates with attached code doesn't require being signed. However, this practice is a good idea given that the administrator knows who the form came from and that he or she is authorized to ask for the deployment of the form.

The process for publishing an administrative form isn't particularly difficult; however, these forms do require access to central administration to complete—access that only a handful of users will have. After they are published, these administrator-approved form templates can be activated in any site collection on any web application in the farm.

Deciding how to add code to your InfoPath form (or for that matter whether it's right to put code in the InfoPath form) isn't as straightforward as it might seem at first. The simple options of script or .NET code rapidly branch into an array of options that leaves no stone unturned. Whether you choose VSTA or VSTO, the 2003 Object Model, or the 2007 Object Model, you'll at least know the basics of what your decision means both in terms of what you can and cannot do.

Additional Resources

- InfoPath Team Blog
- "InfoPath 2007 Document: Developing InfoPath 2007 Managed-Code Solutions" (Microsoft, March 2007).
- InfoPathdev Community Forums

About The Author

Robert Bogue, MCSE (NT4/W2K), MCSA:Security, A+, Network+, Server+, INet+, IT Project+, E-Biz+, CDIA+, is president of Thor Projects LLC, which provides SharePoint Consulting services to clients around the country. He has contributed to more than 100 book projects and numerous other publishing projects. His latest book is The SharePoint Shepherd’s Guide for End Users. (You can find out more about the book at www.SharePointShepherd.com.)

Bogue has been part of the Microsoft Most Valuable Professional (MVP) program for the past 5 years. He was most recently awarded for Microsoft Office SharePoint Server. Before that, Bogue was a Microsoft Commerce Server MVP and Microsoft Windows Servers-Networking MVP.

Bogue runs the SharePoint Users Group of Indiana (SPIN, www.spindiana.com), and he is also a member of the steering committees for the Indiana Windows Users Group and Indianapolis .NET Developer Association. In addition to speaking at local and regional events, Bogue speaks at national conferences. He blogs at www.thorprojects.com/blog , and you can reach him at Rob.Bogue@thorprojects.com .