

Microsoft InfoPath 2003 By Example

Author: [Darshan Singh](#) (Managing Editor, PerfectXML.com)

Last Updated: April 20, 2003

Abstract

On October 9, 2002, Microsoft **announced** a new Office application, code-named "**XDocs**". Built from the ground up to work natively with XML, XDocs enables creation of forms that can be easily integrated with Web services, databases, XML and any XML-enabled system, such as BizTalk Server. XDocs was later **renamed** as **Microsoft InfoPath 2003**.

Enterprises can now easily create rich and dynamic forms using the WYSIWYG interface provided by InfoPath. The native format for data storage in InfoPath is XML, and hence the information gathered using InfoPath forms can be very easily integrated with virtually any backend system that can understand and work with XML.

In this two-part article series, I'll show you how to create a sample InfoPath forms application. This sample application will allow employees to submit TimeOff requests, and managers can then approve/reject the request. The primary goal behind this sample application is to illustrate how easy it is to create a rich InfoPath form that communicates with XML Web services.

Note that this article is based of Office 2003 Beta 2 – things might change in the final release.

To try out the sample application, you'll need the following:

- [Microsoft Windows 2000](#), [Windows XP](#), or [Windows Server 2003](#),
- [Microsoft .NET Framework 1.0 SP2](#) or [higher](#),
- [IIS 5.0](#) or [higher](#); or [ASP.NET Cassini Sample Web Server](#),
- [Microsoft InfoPath 2003](#)
- [Microsoft Access](#)

[Click here](#) if you want to see how the TimeOff 2003 form created in this article would look like.

Introducing InfoPath

Today, in a typical enterprise scenario, varied applications are used to gather the information. For instance, a sales representative might use Microsoft Word to write his sales trip or a performance review document, excel to create estimated and actual sales figures spreadsheets, Outlook form to submit a maintenance request to his systems (IT/MIS) department, some custom application for submitting expense report, some other third party application to store CRM and support details, and so on. This leads to duplication of efforts, difficult to find information, and not being able to reuse the data with backend/orchestrating processes. In addition, the static forms limit the ability to provide required information. In summary, the current information gathering methods make it hard to reuse the data across business processes and require significant development work.

Microsoft created InfoPath to fill this gap – to allow organizations to easily create dynamic and rich forms – and at the same time, allow the gathered information to be easily accessible and usable for application integration. XML, being the neutral data-interchange format, was the best choice to be used for representing the collected data. InfoPath can be used to design and to fill out a form.

Here are some of the features:

- InfoPath provides a WYSIWYG interface to design forms, provides rich set of form controls (such as Rich Text Box, Date Picker, Repeating Section, and so on) and layout options (tables, optional section, and so on).
- Supports Auto-complete when forms are being filled in.
- Data Validation: based on XSD (XML Schemas) and script-based
- Ability to connect to ADO relational data sources (Access and SQL Server), XSD/XML, and Web services.

- Supports creating forms based on custom XSD schemas.
- Supports merging multiple forms into a single form.
- Integrated with SharePoint™ Services
- Conditional Formatting (example: if actual sales is less than estimates sales, then show actual sales field in red, else in green color)
- Ability to work offline.
- Includes more than 20 sample forms that be used out-of the box, or further customized.
- Supports repeating sections and repeating tables for dynamic forms.
- Allows publishing the form to SharePoint forms library, on a Web server, or to a shared folder on a local or network computer.
- Security features includes locking down the form design, and including digital signatures.
- Ability to save filled-in form as .MHT format that can be viewed inside a Web browser
- Bundles Microsoft Script Editor (MSE) to be used for script code (validation, form submission, and other event handling)
- Contains built-in support for UDDI.
- Supports Object Model to be used by developers.

Here are some of the limitations (as of Beta 2): (some of the points below may be *by-design* and not really a limitation)

- Supports only XSD. DTD and XDR not supported.
- No built-in support to authenticate against Active Directory (ADSI).
- Smart Tags not supported.
- No support for Pocket PC/PDA.
- Does not use or support XForms (uses XSLT instead)
- Microsoft Script Editor (MSE) does not support Intellisense.
- Does not support importing scanned versions of existing paper forms.
- InfoPath needs to be installed to fill-in the form
- Only document-literal kind of Web services supported.
- XML file, database, and Web service can be used as a secondary data source only with "Lookup"-type fields such as combo box and list box.

More information on InfoPath can be found [here](#).

TimeOff 2003

As mentioned earlier, the goal of this sample application to illustrate some of the InfoPath 2003 features – specifically the integration with Web services. The TimeOff 2003 sample form application presented in this article has the following components:

- **Microsoft Access database**: To store timeoff requests and other details (can be any other relational database, such as Oracle)
- **DataService.asmx**: An ASP.NET XML Web service that provides data-access Web methods. The form uses this Web service to populate some controls on the form, and also to submit the TimeOff request.
- **Holidays.aspx**: A sample ASP.NET Web page to illustrate how it can be used as the source for Custom task pane with the InfoPath form. Generally, you would create Custom task pane to facilitate some actions (look at UIBASICS.XSN or CD_EDIT.XSN samples shipped with InfoPath for an example of this), but in this example, for simplicity, we'll set an ASPX page as the source for Custom task pane – this .aspx page simply uses a grid to show holidays observed in year 2003.
- **TimeOff2003.xsn**: InfoPath form

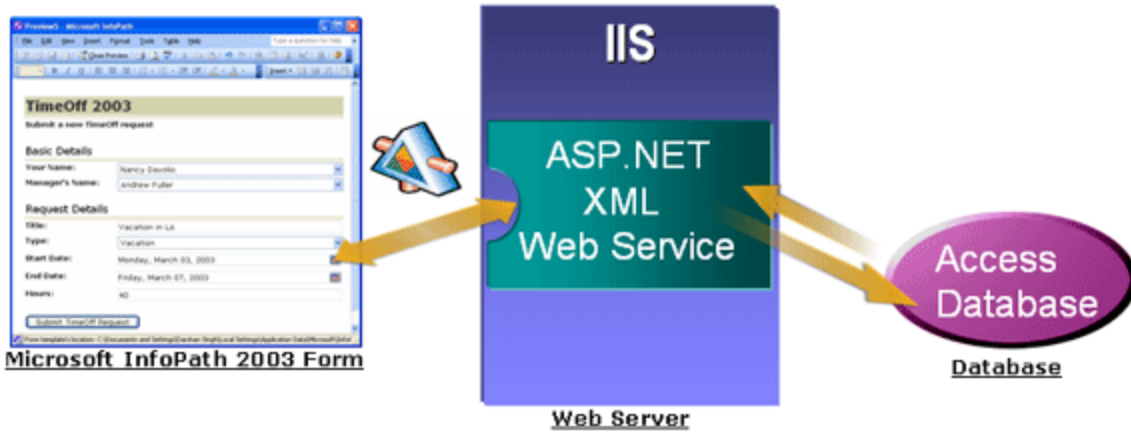


Figure 1: TimeOff 2003 InfoPath form communicating with ASP.NET XML Web service

Download and try out TimeOff 2003:

1. Unzip TimeOff2003.zip: Make sure all the files are unzipped into a folder `c:\PerfectXML\TimeOff` - this path is hard-coded at various places, such as database connection string in the web.config.
2. Create an IIS virtual directory or a Web site named **TimeOff** and point it to `c:\PerfectXML\TimeOff`
3. Make sure Web service is working fine: Open Internet Explorer and browse to `http://localhost/DataService.asmx` and try out Web methods such as `GetRequestTypes`, `GetEmpList`, etc.
4. Browse to `http://localhost/Holidays.aspx` just to make sure the Web page is working fine. You should see a grid showing Holidays observed for year 2003.
5. Open `TimeOff2003.xsn` either in design mode (right click on the form and select Design) or in the data-entry mode (double click on the file).

TimeOff 2003 Database

The sample TimeOff Access database consists of five tables: **tblEmployees** (contains employees details), **tblHolidays** (Holidays Observed), **tblRequestTypes** (types of timeoff requests, such as vacation, sick, etc.), **tblStatusText** (current status of request, such as pending, approved, etc.), and **tblTimeOffRequests** (contains actual timeoff requests details):

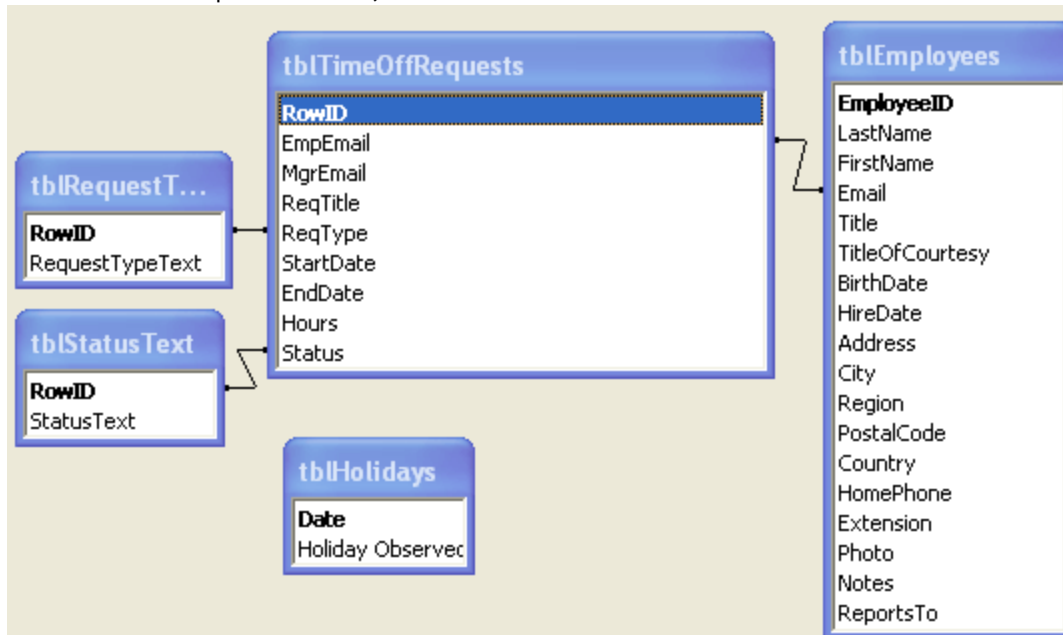


Figure 2: TimeOff 2003 Access Database Schema

ASP.NET XML Web service

Let's now look at an .asmx page – an ASP.NET Web service that connects to the above database (via the connection string provided in the web.config file) and allows reading from and writing into this database. For brevity, we'll write the code directly in the .asmx page instead of creating a code-behind.

The following Web service code initializes the connection string member variable in the constructor by reading the value from web.config. The first three Web methods simply call a supporting private function (**GetXML**) by passing it the table name, columns list, and the where clause. The **GetXML** method then uses **XmlDataDocument** and **OleDbDataAdapter** to run a SELECT query and returns **DocumentElement** property of the **XmlDataDocument** instance. The **InsertRequest** method accepts TimeOff request details as an XML string. It loads this XML string into DOM Document (**XmlDocument**), selects the data values and uses them to insert a new row in the **tblTimeOffRequests** table.

DataService.asmx

```
<%@ WebService Language="C#" Class="TimeOff2003.DataService"%>

using System;
using System.Xml;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

using System.Configuration;

using System.IO;

using System.Data;
using System.Data.OleDb;

namespace TimeOff2003
{
    [WebService (Namespace="http://Office2003/TimeOff2003",
        Description="TimeOff 2003 Data Web service to be used by InfoPath.")]
    public class DataService : WebService
    {
        private string DBConnectionString;

        public DataService()
        {
            DBConnectionString = ConfigurationSettings.AppSettings["DBConnStr"];
        }

        [WebMethod (Description="Get Employees list.")]
        public XmlNode GetEmpList()
        {
            return GetXML("tblEmployees", "FirstName + ' ' + LastName as EmpName, Email", "");
        }

        [WebMethod (Description="Get absence request types.")]
        public XmlNode GetRequestTypes()
        {
            return GetXML("tblRequestTypes", "*", "");
        }

        [WebMethod (Description="Get request status text and codes.")]
        public XmlNode GetStatusText()
    }
}
```

```

{
    return GetXML("tblStatusText","*", "");
}

[WebMethod (Description="Insert a new TimeOff request.")]
public bool InsertRequest(string TimeOffDetailsXML)
{
    bool bRetVal = true;
    try
    {
        DataSet TimeOffRequestsDS = new DataSet();

        OleDbDataAdapter Adapter = new OleDbDataAdapter("SELECT * FROM tblTimeOffRequests WHERE 1 =
            DBConnectionString);

        Adapter.Fill(TimeOffRequestsDS);

        OleDbCommandBuilder cmdBuilder;
        cmdBuilder = new OleDbCommandBuilder(Adapter);

        DataRow newRow = TimeOffRequestsDS.Tables[0].NewRow();

        XmlDocument doc = new XmlDocument();
        doc.LoadXml(TimeOffDetailsXML);

        XmlNamespaceManager nsmgr = new XmlNamespaceManager(doc.NameTable);
        nsmgr.AddNamespace("my",
            "http://schemas.microsoft.com/office/infopath/2003/myXSD/2003-04-17T05:36:32");

        newRow["EmpEmail"] = doc.SelectSingleNode("//my:EmpEmail", nsmgr).InnerText;
        newRow["MgrEmail"] = doc.SelectSingleNode("//my:MgrEmail", nsmgr).InnerText;
        newRow["ReqTitle"] = doc.SelectSingleNode("//my:ReqTitle", nsmgr).InnerText;
        newRow["ReqType"] = Convert.ToInt32(doc.SelectSingleNode("//my:ReqType", nsmgr).InnerText);
        newRow["StartDate"] = doc.SelectSingleNode("//my:StartDate", nsmgr).InnerText;
        newRow["EndDate"] = doc.SelectSingleNode("//my:EndDate", nsmgr).InnerText;
        newRow["Hours"] = Convert.ToInt32(doc.SelectSingleNode("//my:Hours", nsmgr).InnerText);
        newRow["Status"] = 1;

        TimeOffRequestsDS.Tables[0].Rows.Add(newRow);

        Adapter.Update(TimeOffRequestsDS);
    }
    catch(Exception exp)
    {
        bRetVal = false;

        throw(new SoapException(
            "Failed to add a record in tblTimeOffRequests table.\n\n" + exp.ToString(),
            SoapException.ClientFaultCode));
    }

    return bRetVal;
}

private XmlNode GetXML(string TableName, string ColumnList, string WhereClause)
{
    try
    {

```

```

XmlDataDocument XMLDS = new XmlDataDocument();

OleDbDataAdapter DA = new OleDbDataAdapter(
    "SELECT " + ColumnList + " FROM " + TableName + " " + WhereClause,
    DBConnectionString);

DA.Fill(XMLDS.DataSet);

return XMLDS.DocumentElement;
}
catch(Exception exp)
{
    throw(new SoapException(
        "Error while accessing " + TableName + ". \n" + exp.ToString() + "\n\n",
        SoapException.ClientFaultCode));
}
}
}
}

```

If you browse to the above .asmx Web service page and try out provided Web methods, you should see screens similar to:

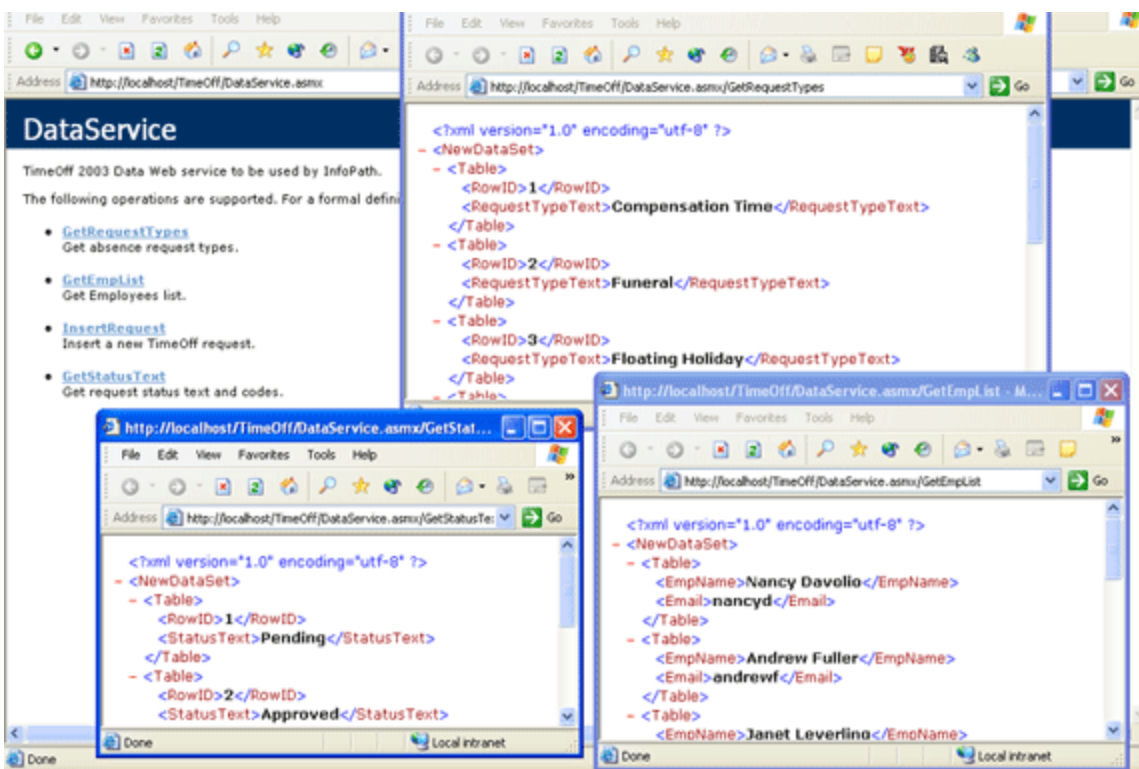


Figure 3: DataService.asmx ASP.NET XML Web service

ASP.NET Web page

Let's now look at an .aspx Web page that will be shown on the custom task pane in our TimeOff 2003 form. Once again, we'll not create the code behind file, but write the code inline the .aspx page.

The following ASP.NET page contains a data grid that is bound to the rows from the tblHolidays table in the Page_Load method. The Web service earlier was written using C# - let's use VB.NET this time:

Holidays.aspx

```

<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
<%@ Import Namespace="System.Configuration" %>
<html>

<script language="VB" runat="server">
Sub Page_Load(Sender As Object, E As EventArgs)

    Dim DS As DataSet
    Dim MyConnection As OleDbConnection
    Dim MyCommand As OleDbDataAdapter

    MyConnection = New OleDbConnection(ConfigurationSettings.AppSettings("DBConnStr"))

    MyCommand = New OleDbDataAdapter("SELECT [Date], [Holiday Observed] FROM [tblHolidays] " & _
        "ORDER BY [ROWID] ASC", _
        MyConnection)

    DS = New DataSet()
    MyCommand.Fill(DS, "Holidays")

    MyGrid.DataSource = DS.Tables("Holidays").DefaultView
    MyGrid.DataBind()
End Sub
</script>

<body topmargin="0" leftmargin="0" marginwidth="0" marginheight="0">
    <ASP:DataGrid id="MyGrid" runat="server"
        BorderStyle="None"
        CellSpacing="1"
        BorderWidth="0px"
        BackColor="#999999"
        CellPadding="2">
        <ItemStyle ForeColor="#000000" BackColor="#FFFFFF"
            Font-Name="Verdana" Font-Size="11px"/>
        <HeaderStyle Font-Bold="True" ForeColor="#000000"
            Font-Name="Verdana" Font-Size="12px" BackColor="#EEEEEE" />
    </ASP:DataGrid>
</body>
</html>

```

The above ASP.NET page produces the following results:

Date	Holiday Observed
1/1/2003	New Years Day
5/26/2003	Memorial Day
7/4/2003	Independence Day
9/1/2003	Labor Day
11/27/2003	Thanksgiving Day
11/28/2003	Day after Thanksgiving
12/24/2003	Christmas Eve Day
12/25/2003	Christmas Day
12/31/2003	New Year's Eve Day

Figure 4: Holidays.aspx ASP.NET Web page to be displayed on the custom task pane

We now have the basic framework ready. Let's now design the actual TimeOff form using InfoPath.

Designing a Form using InfoPath

Here are the steps to create a form used to submit a new TimeOff request:

Step 1: Start Microsoft InfoPath 2003, and click on **File | Design a Form**, and then select **New Blank Form** from the Task Pane.

Step 2: Click on **Layout** on the "Design Tasks" pane, drag and drop "Table with Title" layout onto the form. Change the title to "TimeOff 2003", increase the font-size to 18 and make it bold; and update the description on the next row to "Submit a new TimeOff request".

Step 3: Next, insert a two column table, having 7 rows, with captions on the left column as "Your Name:", "Manager's Name:", "Request Title:", "Type:", "Start Date:", "End Date:", and "Hours:".

Step 4: Click on **Controls** link on the task pane, drag and drop combo boxes (drop-down list boxes) in front of "Your Name:", "Manager's Name:" and "Type:" columns; Text boxes in front of "Request Title:" and "Hours:"; and Date Picker controls in front of "Start Date:" and "End Date:". Finally, drag and drop a button control on the form.

By now your form should look like:

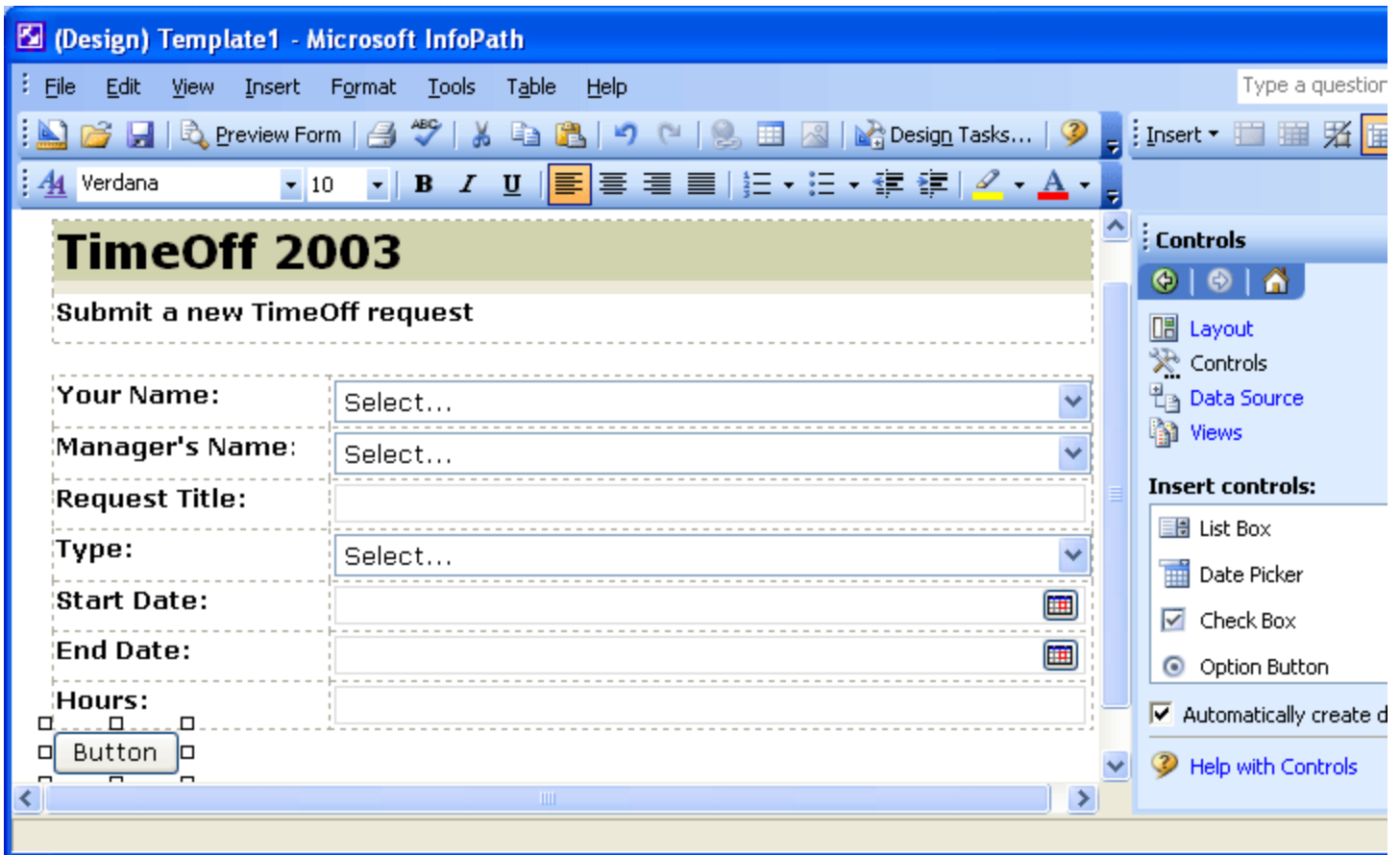


Figure 5: TimeOff form being designed - after few initial steps

Step 5: Let's now set the control properties. Right click on the combo box (drop-down list) control in front of "Your Name" and select "Drop-Down List Box Properties...". Change the field name to **EmpEmail**, check "Cannot be blank", Click "Lookup in a database, Web service, or file" radio button and then click on **Secondary Data Source...** button.

Click "Add" button on the "Secondary Data Source..." dialog, select "**Web service**" radio button, click Next, and type **http://localhost/TimeOff/DataService.asmx?WSDL** as the Web service URL. This is the place where you can query UDDI and link to a Web service. In our case, we know that we would like to connect to DataService.asmx. Click Next, select **GetEmpList** from the list of Web methods, click Next thrice, and then on the Finish button. Close the Secondary Data Source dialog box.

Data Source is now defined, we need to link the combo box by specifying the XPath expression. Click on the **XPath** button next to the **Entries:** text box on the Drop-Down List Box Properties dialog box, and select Table from the hierarchy:

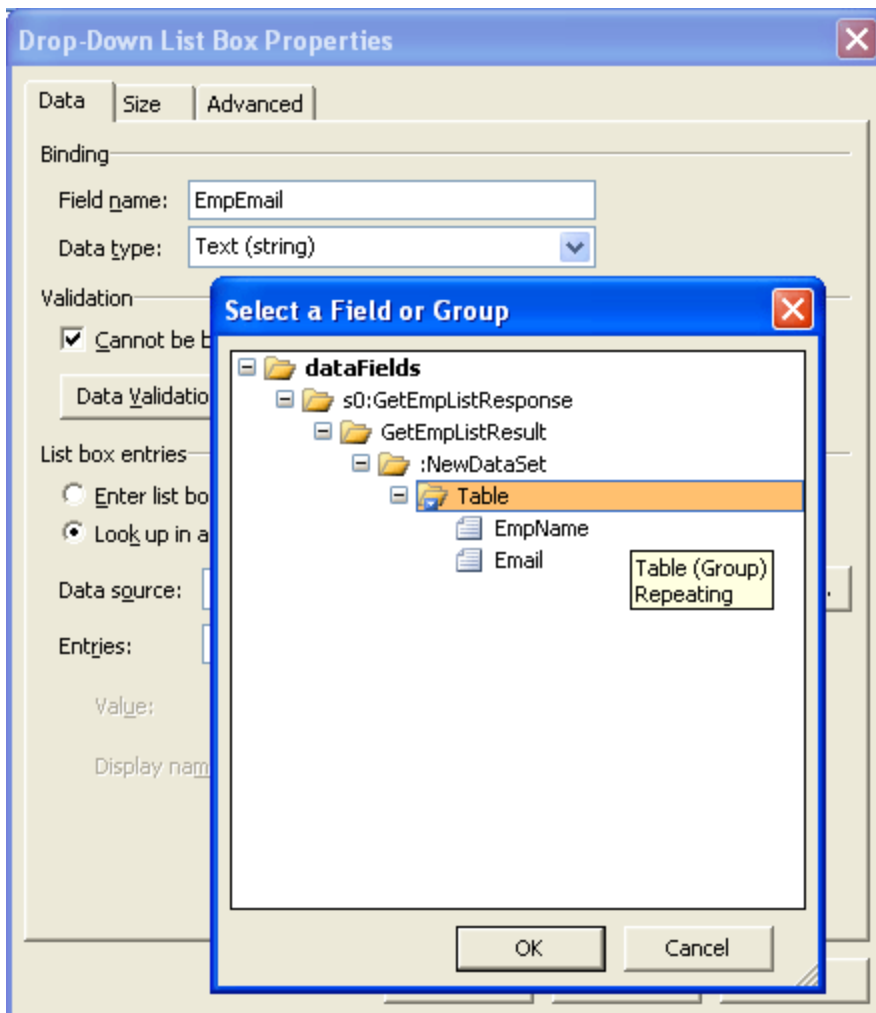


Figure 6: Binding the Drop-Down list box (using XPath) with the XML returned by the Web method.

Change the **Value** to **Email** (instead of EmpName) and click OK.

Step 6: Repeat the step 5 for the drop-down list boxes next to "Manager's Name" and "Type" - name the fields as **MgrEmail** and **ReqType**. It is important that you specify these field names, as they are used in the Web service when a new timeoff request is added (InsertRequest Web method). You can Bind MgrEmail with the existing data source (GetEmpList). But create a new data source for ReqType combo box and bind it to **GetRequestTypes** Web method, choose Value as "RowID" and Display Name as "RequestTypeText".

Right click on text box control next to "Request Title" and select "Text Box Properties". Name the field as **ReqTitle**, check "Cannot be blank" and specify 'TimeOff Request Title such as "Vacation in Florida"' as the *Placeholder* value (under the Display tab on the Properties dialog box). Click OK.

Name the two date picker controls as "StartDate" and "EndDate" respectively, check "Cannot be blank". For End Date control, click on "Data Validation..." button on the control properties dialog box, click Add..., and specify the validation condition that end date cannot be less than start date.

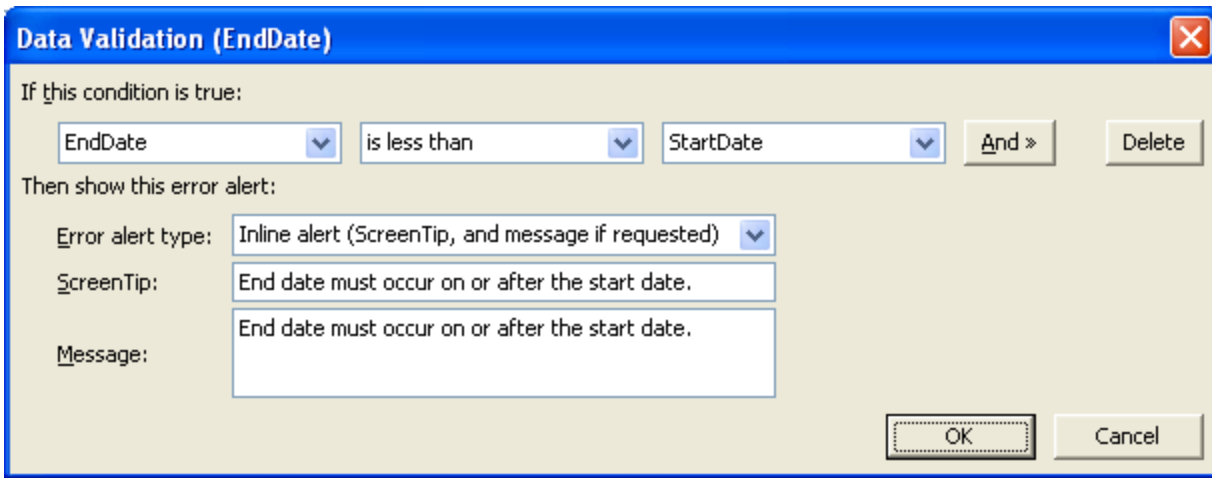


Figure 7: Making sure End date is not less than Start date.

Next, click on text box next to "Hours"; name the text box control as "Hours", select the data type as Whole Number (integer) and check "Cannot be blank".

Click on the button and select **Button Properties**, select the action as **"Submit"**, the "Submitting Forms" dialog box pops up, select "Enable Submit" and then select **Submit using custom script**. Change the submit menu item caption value to **Su&bmit TimeOff Request**; Click on **Submit Options** button, select **Close the form**, and check "Instead of default message, show custom message" and write "Your TimeOff request was submitted successfully!" for success and "Failed while submitting the request. Please contact darshans." as the failure message.

Make sure **Open Microsoft Script Editor** is checked and then click on OK button. InfoPath launches Microsoft Script Editor, and allows us to write code under `function XDocument::OnSubmitRequest(eventObj)`.

We'll be writing the custom JScript code that uses MSXML 3.0 XMLHTTP object to POST the timeoff details to the DataService.asmx Web service.

Write the following code under the `OnSubmitRequest` function:

```
try
{
    var objXH = new ActiveXObject("MSXML2.XMLHTTP.3.0");
}
catch(exception)
{
    XDocument.UI.Alert("Could not create MSXML2.XMLHTTP.3.0 object.\r\n" +
        exception.number + " - " + exception.description);
    eventObj.ReturnStatus = false;
    return;
}

try
{
    objXH.open("POST", "http://localhost/TimeOff/DataService.asmx/InsertRequest", false);
    objXH.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    strPostXMLText = XDocument.DOM.xml;

    strPostXMLText = strPostXMLText.replace(/\+/g, "%2B");
    strPostXMLText = strPostXMLText.replace(/\&/g, "%26");

    //XDocument.UI.Alert(strPostXMLText);

    objXH.send("TimeOffDetailsXML=" + strPostXMLText);
}
```

```

if(objXH.status != 200)
{
    XDocument.UI.Alert("Failed while sending the request.\r\n" + objXH.status + " - " +
        objXH.statusText);

    eventObj.ReturnStatus = false;
    return;
}
}
catch(exception)
{
    XDocument.UI.Alert("Failed while sending the request.\r\n" + exception.number + " - " +
        exception.description);
    eventObj.ReturnStatus = false;
    return;
}

eventObj.ReturnStatus = true;
return;

```

Save the script. Our form is now ready to be tested.

Step 8: Save the form as **TimeOff2003.xsn** under the C:\PerfectXML\TimeOff folder.

Step 9: Click on the **Preview Form** button, and you'll notice how the combo boxes are filled in; verify that validation is working fine, fill in all the fields and then click on "Submit TimeOff Request" button. If the submit fails, make sure the namespace used by the form and the one used in the Web service match. In the design view, click on **Data Source** link on the Task pane, right click on **myFields** and select Properties. Make sure that the namespace specified under the details tab match the namespace used the *InsertRequest* Web method.

Once the request is submitted successfully, open the Access database and you should see a new row in the **tblTimeOffRequests** table for the new request that you just submitted.

The final step is to show 2003 holidays in the task pane. To do this, open the form in the design mode, click Tools | Form Options..., click on Advanced tab, check Enable Custom task pane, type "2003 Holidays" as the task pane name, and **http://localhost/TimeOff/Holidays.aspx** as the location:

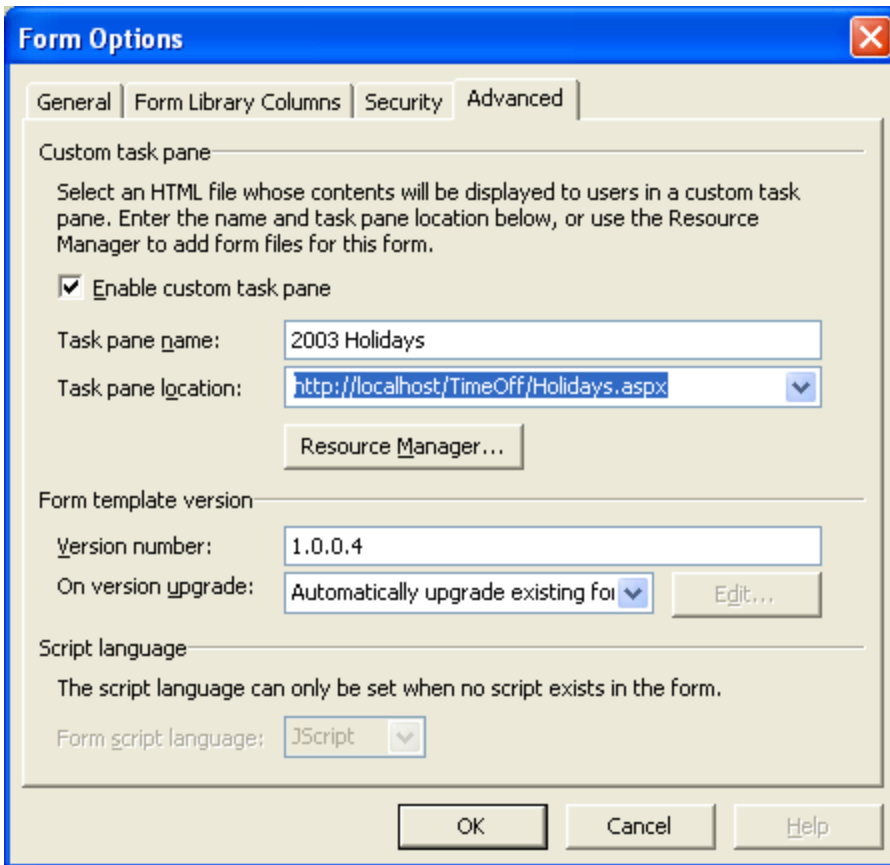


Figure 8: 2003 Holidays on Custom Task Pane.

Here is the final form in the preview mode; the built-in validation makes sure correct data is posted, also notice the ASP.NET Web page (2003 holidays) being displayed in the custom task pane:

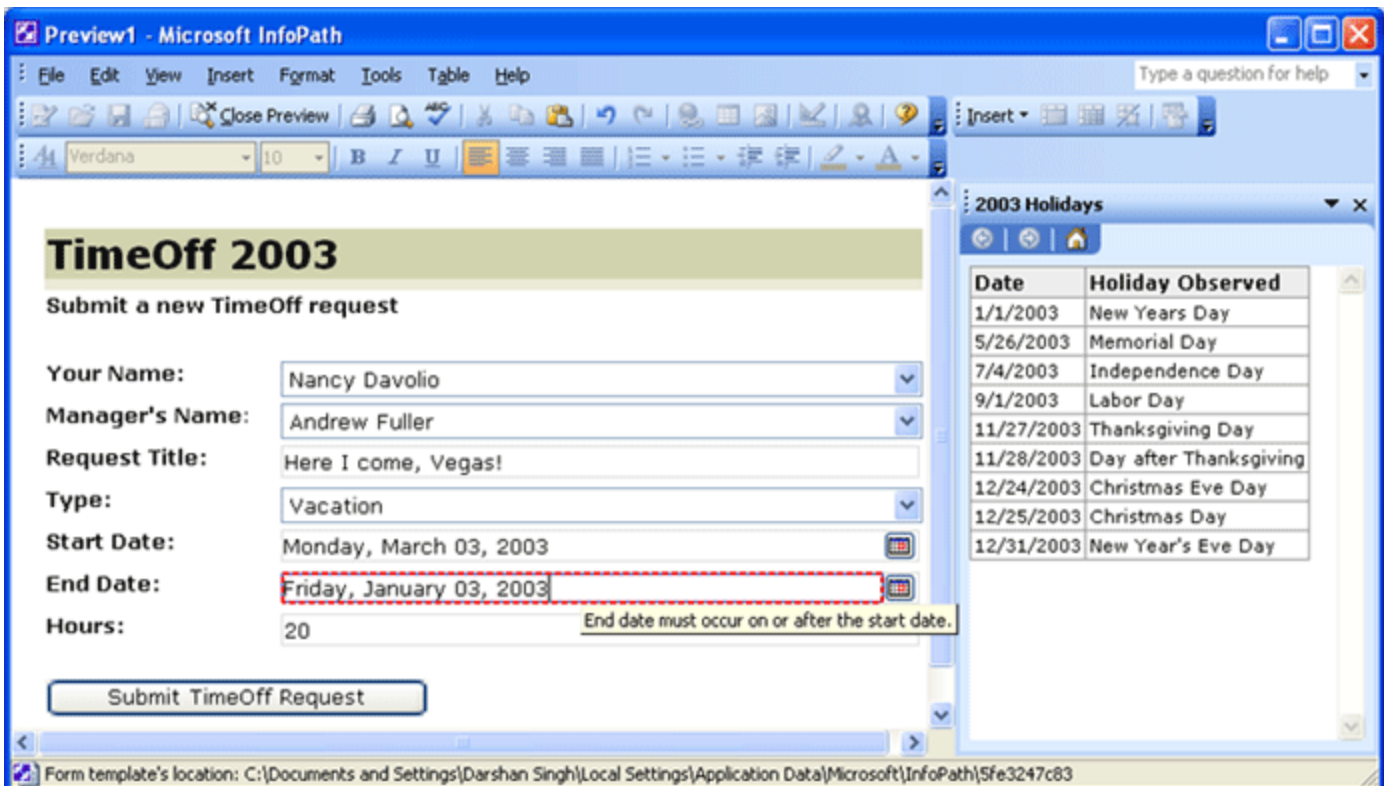


Figure 9: Fields Validation makes sure correct data is submitted.

Before I end this article, here are some tips/facts that I think would be useful:

Tips

- You can use regular expressions in your script code for advanced validation. For example, following validation script code ensures that correct phone number (ex: 111-111-1111) is entered:

```
function mssoxd_my_ClientPhone::OnValidate(eventObj)
{
  if(eventObj.Site.nodeTypeValue != "")
  {
    var strPhone = eventObj.Site.nodeTypeValue;
    var RegEx = /^(\\(?:\d{3}\) ?|\d{3}-)\d{3}-\d{4}$/g;
    if(!RegEx.test(strPhone))
      eventObj.ReportError(eventObj.Site, "Invalid Phone Number.", false);
  }
}
```

- Even though InfoPath uses XSN as the file extension for form templates, these files are essentially CAB files that you can for instance open with WinZip and extract it to a folder. It consists of bunch of XML files, XSLT stylesheet, XSD schema file, script file, and **manifest.xsf**.

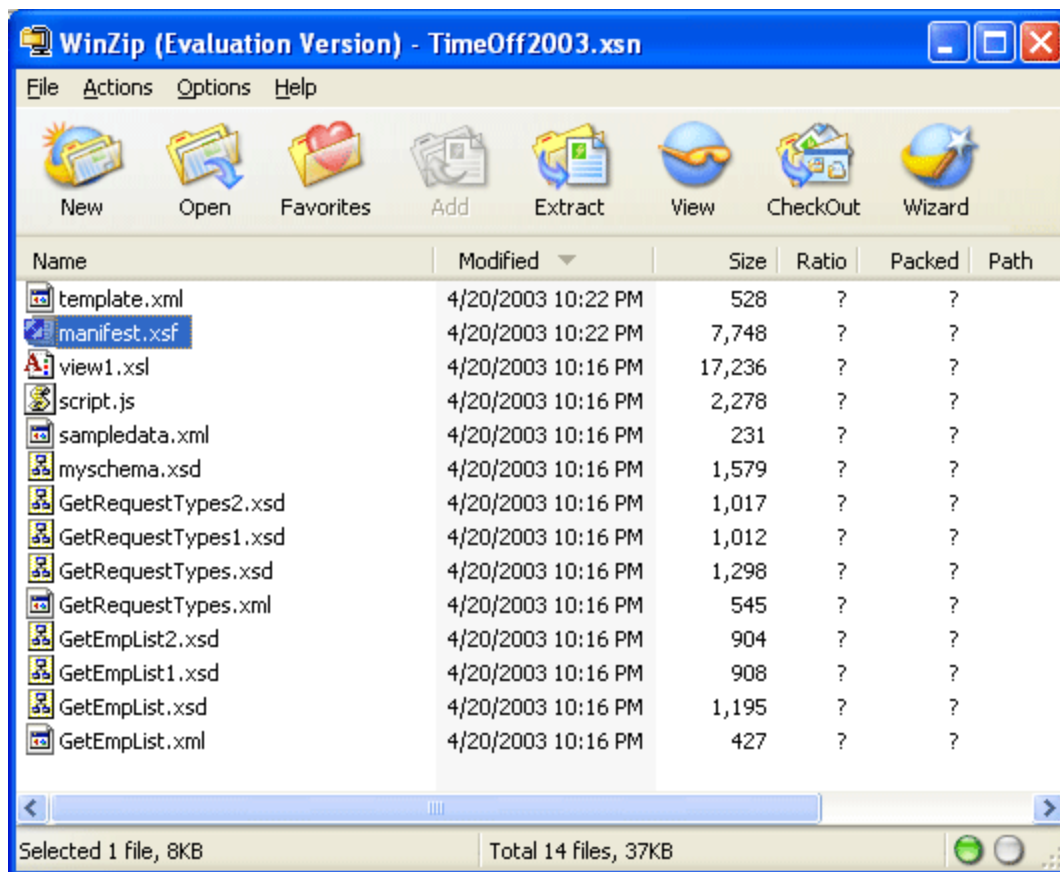


Figure 10: InfoPath Form Template File (XSN) is a CAB file - opened using WinZip

For instance, if you wanted to update an InfoPath form, one option is to load the form in design mode and use File | Save As menu item; alternatively, you can unzip the files into a folder, update the files, create a text file containing list of files (with one filename on each line, enclosed in double quotes if contains spaces), and then run **makecab** command line utility to create a cab file – finally simply change the file extension from cab to xsn.

- InfoPath supports/makes use of following standards:
 - XML 1.0 SE, Namespaces in XML
 - XSD 1.0

- XSLT 1.0
 - XHTML 1.0
 - DOM 1.0
 - XML DSIG
 - SOAP 1.1, UDDI 1.0, WSDL 1.1
- If you setup The Access Control List (ACL) on the form file, InfoPath would respect that and protect the form from unauthorized access.
 - If you want to disable the form design mode, two options include:
 - **Option 1:** Click on Tools | Form Options, and check **Enable Protection** under General tab.
 - **Option 2:** First create a new sub-key called **Designer** under HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\InfoPath, and then under this sub-key, create a new **DWORD** value named **DisableDesigner** and set it's value as **1**. Assuming that user does not have access to system registry, this approach facilitates complete designer lockdown. If user tries to open the file in design mode, following message box is shown:

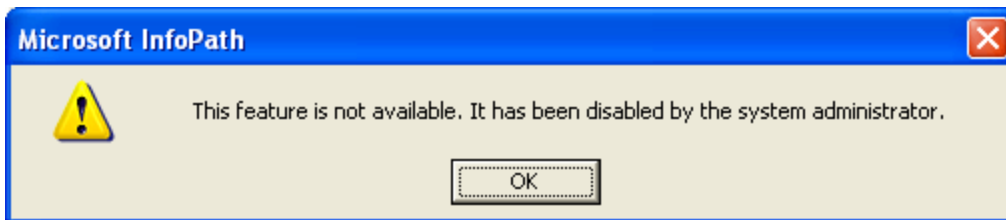


Figure 11: Designer Lockdown

- Microsoft InfoPath team recently posted a ZIP file on **microsoft.public.infopath** on msnews.microsoft.com containing Advanced InfoPath samples. I highly recommend looking at those samples.

Summary

Microsoft InfoPath 2003, the new member application in the Office 2003 suite, facilitates easily creating rich and dynamic forms, which can be used to gather the information. And because the data format used is XML, the collected information can be quickly re-used and integrated with other business processes.

In this article, you learned how to create an InfoPath 2003 form that pulls data from a Web service, and uses script to post a form to a Web service method. In addition, I also illustrated some other InfoPath tips and techniques.

Part 2 of this article will highlight integrating with SQL Server 2000 (using SQLXML) and BizTalk Server, and I'll also detail the manifest.xsf and other files contained in the InfoPath form template (xsn file).

Resources

- [InfoPath Home Page](#)
- [InfoPath Technical FAQ](#)
- [Using InfoPath to Create Smart Forms](#)
- [XDocs / InfoPath News](#)