TopLeaf 7 How To:

Server Applications with ASP and XMLComposer

1. Introduction

TopLeaf is a highly adaptable XML composition engine that can be used as part of larger enterprise solutions where dynamic document generation is required. By accessing TopLeaf through the API, you can build custom interfaces to TopLeaf, for either general access to TopLeaf's features through a custom user interface, or you can build integrated solutions that simply automate production of composed documents.

For dynamic document creation, you need to understand the process flow in TopLeaf, as well as how to program using API calls through one of the three available API interfaces (command-line, DLL, or ActiveX scripting object). Alternatively, you can choose Metaformix's ***XMLComposer*** to take care of the TopLeaf issues so you can focus on the more important overall application design.

In this article, we'll look at how to build a simple Active Server Pages (ASP) web application that uses ***XMLComposer*** to create formatted PDFs from web form data merged with a document template. The same approach applies equally well to Cold Fusion pages, PHP, or other web scripting languages.

2. Metaformix News Release Builder

At Metaformix, we built an ASP application to generate our news releases so they can be produced and released onto our web site quickly, with very little input from the user. This makes for a light-weight, multi-tiered design that is accessible on our intranet. Like any document application, the hard part is settling on a document layout. We're not going to address the document layout in this article. Instead, we'll focus on two things:

- The ASP application itself
- Template-based documents in TopLeaf

The ASP Application

The Metaformix News Builder is a very straightforward Active Server Pages application that does two things: Accept input from the user, and write it to a text file as XML-formatted data.

Once the data is written to the file, XMLComposer manages the passing of the job through TopLeaf to generate the formatted PDF.

The structure of the ASP end of things is simple. We need a form to welcome the user and confirm this is what they want to be doing, another to permit the user to fill in the various fields of the news release, another to confirm that the release is to be generated, and we need a final one to direct them to the output.

This could all be done in one massive form, but for ease of set-up and maintenance, we chose to break it up into its functional bits.

After building the basic layout and style sheet, we set up the web site. We're using IIS, so we create a virtual directory for the application, and then add the component files as follows.

global.asa: This file defines application defaults and global variables.

```
'=======================
'News builder global.asa, placed in application's root directory
'=======================
<SCRIPT LANGUAGE=VBScript RUNAT=Server>


Sub Application_OnStart
 'specify the XMLComposer watched folder for News publishing
 Application("InputDir")="d:\TopLeaf\MxNews\Input\"

 'specify the output folder for all generated docs
 Application("OutputDir")="c:\inetpub\MxNews\output"


 'the name of the TopLeaf partition where data is assembled
 Application("XMLFile") = "MxNews"

 'Default application name
 Application("AppName") = "Metaformix News Release Builder"

End Sub

</SCRIPT>
```

default.asp: This file is the "entry" page into the application. The purpose is simply to present the user with an initial screen, from which they can choose to begin creating a news release.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Default.asp -->
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
<title>Metaformix News Release Builder</title>
<link rel="stylesheet" type="text/css" href="news.css">
</head>
<div id="main-title">News Release Builder</div>
<!-- Just a container for the image that runs across the top of the page -->
<div id="header-image"></div>
<div id="navbar">
    <a href="default.asp" class="current">home</a> ||
    <a href="MxNewsInput.asp">start</a>
</div>
<div id="headline">Metaformix News Release Builder</div>
<div id="main-text">
<body>
<h1>Welcome To the<br>TopLeaf Active Server Pages</h1>
<p>To proceed to the news release builder, click here:
<FORM Name=nextpg METHOD=POST ACTION=MxNewsInput.asp>
<input type=submit name=btnSubmitdefault value="Create News Item">
</FORM></p>
<div id="footer">
Copyright © 2005 - Metaformix Information Systems Inc</div>
</body>
</div>
</html>
```

MxNewsInput.asp: This page generates the input fields for the news item. A headline, release date, lead-in paragraph, news body, and alternate contact info are provided.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
<title><%=Application("AppName")%> - News Release Builder</title>
<link rel="stylesheet" type="text/css" href="downtown.css" />
</head>

<div id="main-title">News Release Builder</div>
<div id="header-image2"></div>
<div id="navbar">
    <a href="default.asp">home</a> ||
    <a href="MxNewsInput.asp"  class="current">start</a>
</div>
<div id="headline">News Release Builder</div>
<div id="main-text">
<body>
<table border=0 cellpadding=0 cellspacing=0>
<tr><td align="center" bgcolor=#DDDDDD border=1>
<b>Gather Data</b></td></tr>
<tr><td><p>Enter the data for the news release.</p>
</td></tr>
<tr><td>
<FORM ACTION=MxNewsValidate.asp METHOD=POST>
```

```
<p>All fields are REQUIRED, except where indicated</p>
<table>
<tr><td><p>Headline:  </td>
<td><input type=text name=Headline size=40 value=""></td></tr>
<tr><td><p>Release Date:  </td>
<td><input type=text name=Date size=40 value=""></td></tr>
<tr><td><p>Lead Paragraph: </td><td>
<textarea name="Lead" wrap=virtual cols="50" rows="10"></textarea></td></tr>
<tr><td><p>Body: </td>
<td><textarea name="Body" wrap=virtual cols="50"
rows="10"></textarea></td></tr>
<tr><td><p>Alternate contact info (non-Metaformix, optional): </td>
<td><input type=text name=Altcontact value="" size=40></td></tr>
<tr><td><p> </td><td><input type=submit value="Next >>"></td></tr>
</table>
</td></tr></table>
</FORM>
</td></tr></table>
</body>
<div id="footer">
Copyright © 2005 - Metaformix Information Systems Inc.</div>
</div>
</html>
```

MxNewsValidate.asp: This page checks that the user input is complete enough to proceed with generating the news release. It also gives the user a chance to back out of the process. If it does validate, then the user can submit the data for processing. If it does not validate, the user must go back and provide complete information. In this application, all required fields must have something in them — that's the only requirement for validation.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- MxNewsValidate.asp: check input -->
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
<title><%=Application("AppName")%> - Check and Save the Input</title>
<link rel="stylesheet" type="text/css" href="downtown.css" />
</head>

<div id="main-title">News Builder</div>
<!-- Just a container for the image that runs across the top of the page -->
<div id="header-image4"></div>
<div id="navbar">
    <a href="default.asp">home</a> ||
    <a href="TLDemoDBQuery1.asp"  class="current">start</a>
</div>
<div id="headline">News Builder</div>
<div id="main-text">
<body>
<h2>Check and Save the Input</h2>
<%

'this function deals with any quotes in the input.
'the tricky bit is passing a tag in that has an attribute.
'for simplicity, we just swap " with ' in that case.

Function QuoteCheck( sString )
    Dim ct
    Dim sOut
    Dim sTest
    Dim inTag
```

```
    ct = 1
    inTag = FALSE

    While ct < Len(sString)+1
        sTest = Mid(sString,ct,1)

        select case sTest
            case "<"
                inTag = TRUE
                sOut = sOut & sTest
            case ">"
                inTag = FALSE
                sOut = sOut & sTest
            case Chr(34)
                if not inTag then
                    sOut = sOut & "&quot;"
                else
                    sOut = sOut & "'"
                end if
            case else
                sOut = sOut & sTest
        end select
        ct = ct + 1
    Wend
    QuoteCheck = sOut

End Function

    'user data input from the previous form
    Dim userHeadline
    Dim userDate
    Dim userLead
    Dim userBody
    Dim userAltcontact
    Dim bSkipInsert
    Dim sqlStatement

    On Error Resume Next

    userHeadline = QuoteCheck(Request.Form("Headline"))
    userDate = QuoteCheck(Request.Form("Date"))
    userLead = QuoteCheck(Request.Form("Lead"))
    userBody = QuoteCheck(Request.Form("Body"))
    userAltcontact = QuoteCheck(Request.Form("Altcontact"))

    bSkipInsert = FALSE

    if (Len(userHeadline)=0 OR Len(userDate)= 0 OR _
       Len(userLead)=0) then
        bSkipInsert = TRUE
    end if

    if bSkipInsert then
%>
<p>Sorry, some required values were missing. Make sure you have a Headline,
Date and Lead.</p>
<%
    end if

    if Not(bSkipInsert) then

         'do the insert
        On Error Resume Next
```

```
%>
<p>The data you entered has been accepted.</p>

<FORM ACTION=MxNewsPublish.asp Method=POST>
<p align=center>To publish the document, click the "Publish" button.
<input type=hidden name=Headline value="<%=userHeadline%>" >
<input type=hidden name=Date value="<%=userDate%>" >
<input type=hidden name=Lead value="<%=userLead%>" >
<input type=hidden name=Body value="<%=userBody%>" >
<input type=hidden name=Altcontact value="<%=userAltcontact%>" >
<input type=submit value="Publish">
</p>
</FORM>
<%
    end if
%>
<FORM ACTION=TLDemoDBQuery1.asp>
<p>To start over, click here: <input type=submit value="Start over"></FORM>
<div id="footer">
Copyright © 2005 - Metaformix Information Systems Inc.</div>
</body>
</div>
</html>
```

MxNewsPublish.asp: This page submits the data to XMLComposer by writing the request as a small XML-tagged file, and also writes an XMLComposer job control file. The job control file is an "ini" file format file that specifies job processing options for XMLComposer. The main purpose of the job control file in this instance is to generate a PDF that has the release date as its name. This allows us to use a unique name, such as a session ID, for the job control file, but a common output file name for the result. (More on the XMLComposer set-up later.)

```


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- MxNewsPublish.asp: publish the document -->
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
<title><%=Application("AppName")%> - Publish News Release</title>
<link rel="stylesheet" type="text/css" href="downtown.css" />
</head>

<div id="main-title">News Builder</div>
<!-- Just a container for the image that runs across the top of the page -->
<div id="header-image5"></div>
<div id="navbar">
    <a href="default.asp">home</a> ||
    <a href="MxNewsInput.asp"  class="current">start</a>
</div>
<div id="headline">News Builder</div>
<div id="main-text">
<body>
<p>
<table><tr><td bgcolor=DDDDDD><b>
Publish Result</b></td></tr>
<tr><td>The generated document will be named
<b><%=Request.Form("Date")%></b>.</td></tr>
<tr><td>The "Go to list of built documents" button lists the documents in
the news output folder.
Depending on server activity, there may be a lag before the TopLeaf-
generated PDF
is listed. The list of documents refreshes automatically every 15
seconds.</td></tr>
```

```
</table>
<%
    Dim fs 'file system object
    Dim sInputDir
    Dim sOutputName
    Dim sQuote

    sInputDir = Application("InputDir")

    sOutputName = Session.SessionID

    sQuote = Chr(34)

    set fs=Server.CreateObject("Scripting.FileSystemObject")
    if fs.FileExists(sInputDir & Application("XMLFile")&".xm") then
        fs.DeleteFile sInputDir&Application("XMLFile")&".xm"
    end if

    Session.Timeout = 1

    'write the xml file. note that it is .xm; this prevents it from being
    'immediately processed by XMLComposer

    Set tf = fs.CreateTextFile(sInputDir&Application("XMLFile") _
         & ".xm", True)
       tf.WriteLine("<data>")
       tf.WriteLine("<row ")
       tf.WriteLine("Headline=" & sQuote & Request.Form("Headline") _
        & sQuote & " ")
       tf.WriteLine("Date=" & sQuote & Request.Form("Date") _
        & sQuote & " ")
       tf.WriteLine("Altcontact=" & sQuote & Request.Form("Altcontact") _
        & sQuote & " ")
    tf.WriteLine(">")
       tf.WriteLine("<Lead>" & Request.Form("Lead") _
        & "</Lead> ")
       tf.WriteLine("<Body>" & Request.Form("Body") _
        & "</Body> ")
       tf.WriteLine("</row>")
       tf.WriteLine("</data>")
    tf.Close

    'write the .xcf job control file
    Set tf = fs.CreateTextFile(sInputDir & sOutputName & ".xcf", True)
       tf.WriteLine("[General]")
       tf.WriteLine("SourceFile=" & sInputDir & Application("XMLFile") _
        & ".xm")
       tf.WriteLine("Name=" & Application("XMLFile"))
       tf.WriteLine("[Workflow]")
       tf.WriteLine("Name=" & Request.Form("Date"))
       tf.WriteLine("[Indicators]")
       tf.WriteLine("Z=News")
    tf.Close

%>
<p>The data has been saved to a file.</p>
<form name=PublishDone ACTION=MxNewsInput.asp>
<input type=submit value="Start Again">
</form>
<form name=PublishDone ACTION=ViewOutput.asp>
<input type=submit value="Go to list of built documents">
</form>
</p>
<div id="footer">
Copyright © 2005 - Metaformix Information Systems Inc.</div>
```

```
</body>
</div>
</html>
```

A few comments on MxNewsPublish.asp might be in order. The XML file being written would look something like:

```
<data>
<row>
Headline="PRESENTING THE NEWS RELEASE BUILDER"
Date="21 Aug 2005"
Altcontact="Jimmy James"
>
<Body>
The body of the news release goes here
</Body>
<Lead>
The lead paragraph of the news release goes here
</Lead>
</row>
</data>
```

The format of this record is pretty arbitrary, and shows that you can pass data as attributes to the "row" element, or as child elements. There is an implication for TopLeaf. If you pass data as attributes, any mark-up in the attribute value will be tossed before it is composed. But, if data is passed as elements, then the mark-up can be preserved. In the case of the news release, we've allowed for lead paragraphs and the body of the news release to contain markup such as hyperlinks or other formatting tags such as "
" by passing these as child elements to the row.

(In ADO applications, you don't have the option to pass data as child elements without additional transformation, because the data record is always presented as a "z:row" element with each column represented by an attribute.)

The written file constitutes the dynamic, or variable data, content that will be merged with the TopLeaf template document.

The .xcf file is the XMLComposer job file. XMLComposer allows processing options to be specified in either, or both, of two ways: through an xmlcompose.ini file which sits in the folder being watched by XMLComposer, and/or through a job control file. Job control files allow additional job-specific options that override default behaviors in XMLComposer, such as naming output files differently from the default, which will either be the name of the XML file being passed in, or the name of the .xcf file being processed.

The name of the XML file is the name of the partition in TopLeaf where the data to go. By doing it this way, we can create multiple PDFs with different content through the same template document in TopLeaf, thus simplifying the overall process. If we didn't do it this way, we would end up with a new TopLeaf partition for each news release request. We're not that interested in preserving the data once the PDF is generated, so we simply pass new data into the partition each time a news release is requested. If we were interested, we would back-end this application with a database to store the information, from which we could later regenerate any news release.

Another problem with this approach is concurrent access by multiple users. XMLComposer serializes requests on a first-in-first-out basis. But if more than one person is requesting a news release at a time, the data submitted first would be overwritten by the second request. If they both named the news release according to the same date, only the second would survive. In these cases, we would probably want to create new TopLeaf partitions for each news releae

request, or alternatively, add some checking to make sure one isn't already being produced with that name, and provide some form of name uniqueness (such as session ID).

Eventually, a clean-up process to get rid of unwanted partitions would need to be added to the application, or managed through TopLeaf itself.

For now, this isn't a concern to us, as we never have more than one person creating a release at one time.

Because we are not writing the data to a database in this simple application, if we needed to regenerate the document, we would have to re-input all the data. Not ideal, but also not within the scope of this application.

Okay: We've got the data going into XMLComposer's watched folder (as defined in the global.asa file). Let's quickly look at the .xcf file being generated.

```
[General]
SourceFile=d:\TopLeaf\TLDemo\Input\315027787.xm
Name=MxNews
[Workflow]
Name=21 Aug 2005
[Indicators]
Z=News
```

The [General] section defines where the source XML data is located through the SourceFile key. In this case, it is the .xm file just written. The Name key defines the name of the TopLeaf partition where the data will go. Because the .xcf file is named the same as the partition, we don't, strictly speaking, require this line.

The [Workflow] section defines processing instructions for TopLeaf to take when working with the XML file. The Name key tells XMLComposer to instruct TopLeaf to write the PDF output to a file named "21 Aug 2005.pdf". Where the file gets written is defined by the XMLComposer xmlcompose.ini file located in the watched folder. This is also defined in the global.asa file, for use by the ASP application.

The [Indicators] section sets a TopLeaf indicator to a specified value. In this case the "Z" indicator (indicator names are single letter alpha characters, A through Z) is being set to the value "News". This becomes a switch for the TopLeaf partition, and certain stylistic controls are varied based on the value of this indicator, which we'll look at later.

The final component of the ASP application is a page to view the built files.

ViewOutput.asp:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- ViewOutput.asp: list contents of a folder, with hyperlinks -->
<%
'    ViewOutput.asp: list contents of a folder
'
'    Copyright (c) 2005 - Metaformix Information Systems Inc.
'
%>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
<meta http-equiv="refresh" content="15"/>
<title>View Generated Documents</title>
<link rel="stylesheet" type="text/css" href="downtown.css" />
```

```
</head>
<div id="main-title">News Builder</div>
<!-- Just a container for the image that runs across the top of the page -->
<div id="header-image6"></div>
<div id="navbar">
    <a href="default.asp">home</a> ||
    <a href="MxNewsInput.asp"  class="current">start</a>
<!--
    <a href="#">Demo 2</a> ||
-->

</div>
<div id="headline">News Builder Output</div>
<div id="main-text">
<body>
<h2>List of Generated Files</h2>
<%
    Dim fs, fol, fc, s, f1, virt,fn, fd, count
    Dim sOutputDir

    sOutputDir = Application("OutputDir") 'declared in global.asa

    set fs=Server.CreateObject("Scripting.FileSystemObject")

    'sOutputDir should be accessible as a virtual directory
    'to be accessible via a web browser.
    set fol=fs.GetFolder(sOutputDir)
    set fc=fol.Files

    virt="output/"
%>
<table align="center" border=1>
<%
    Dim icount, i

    icount = 0

    'count the number of files in the folder
    For each f1 in fc
        icount=icount+1
    Next
    icount = icount-1

    'Build an array of mark-up to list the files.
    'NB: you could also just loop through the list
    'of files and output them. Doing it this way
    'gives you the option of manipulating (eg. sorting) the
    'array before outputting the list

    Dim farray()
    Dim uDate
    Dim cFDate

    Redim farray(icount)
    count = 0

    For Each f1 in fc
        farray(count) = "<tr><td><A HREF='" + virt + f1.name + "'>" _
            + f1.name + "</A></td><td>" + CStr(f1.DateLastModified) _
            + "</td></tr>"
        count = count + 1
    Next

    For i = 0 to icount
        Response.Write  farray(i)
```

```
    Next

    Response.Write "<tr><td align=center>File</td><td
align=center>Generated</td></tr>"
    i = icount
    while(i >=  0)
        Response.Write  farray(i)
        i = i - 1
    wend
%>
</table>
<form action=MxNewsInput.asp>
<input type=submit value="Start again"/>
</form>
<div id="footer">
Copyright © 2005 - Metaformix Information Systems Inc.</div>
</body>
</div>
</html>
```

ViewOutput.asp just lists the files in name order, and gives a hyperlink to them so that the PDFs can be opened. If sOutputDir is on an externally-accessible web site, then the news release is already web-available. We don't do that, though, as we like to preview the news release before posting externally.

Template-Based Documents in TopLeaf

With the ASP end of things taken care of, we can turn our attention to TopLeaf again. We are sending TopLeaf an XML document that is built from a web form. The content of the web form is unlikely to represent all that we want in the document, so we want to merge that data with a template document resident in TopLeaf.

As with any document in TopLeaf, the first step is to design the overall layout. We do this by creating a new publication (which is where the style information is maintained), and then using sample data to build test partitions to make sure things are working the way we want.

We're skipping the details of doing that in this paper, but we'll focus on a couple of aspects of the mappings for the publication, because this is how a template is built.

Again, there is no particular need for a DTD for the template, but if you have one, it doesn't hurt. In the case of the news release, no DTD is used.

Here is the content of the template document, **newstempl.xml**:

```


<tmpl:Head>
<tmpl:Title>For Immediate Release</tmpl:Title>
<tmpl:Mainhead1><tmpl:Headline/></tmpl:Mainhead1>
<tmpl:Subhead2>Delta, BC - <tmpl:Date/></tmpl:Subhead2>
<tmpl:Simplepara><tmpl:Lead/></tmpl:Simplepara>
<tmpl:Simplepara><tmpl:Body/></tmpl:Simplepara>
<tmpl:Subhead>Contact</tmpl:Subhead>
<table border="0" name="contact">
<tr>
<td><tmpl:logo/>
<tmpl:Tagline>Go Beyond</tmpl:Tagline>
</td>
<td>
<tmpl:Simplepara>
Tel: 604-943-3835<tmpl:br/>
Fax: 604-943-6746<tmpl:br/>
Web: <a
```

```
href="http://www.metaformixis.com">http://www.metaformixis.com</a><tmpl:br/>
Email: info@metaformixis.com</tmpl:Simplepara>
<tmpl:Simplepara><tmpl:Altcontact/></tmpl:Simplepara>
</td>
</tr>
</table>
</tmpl:Head>
```

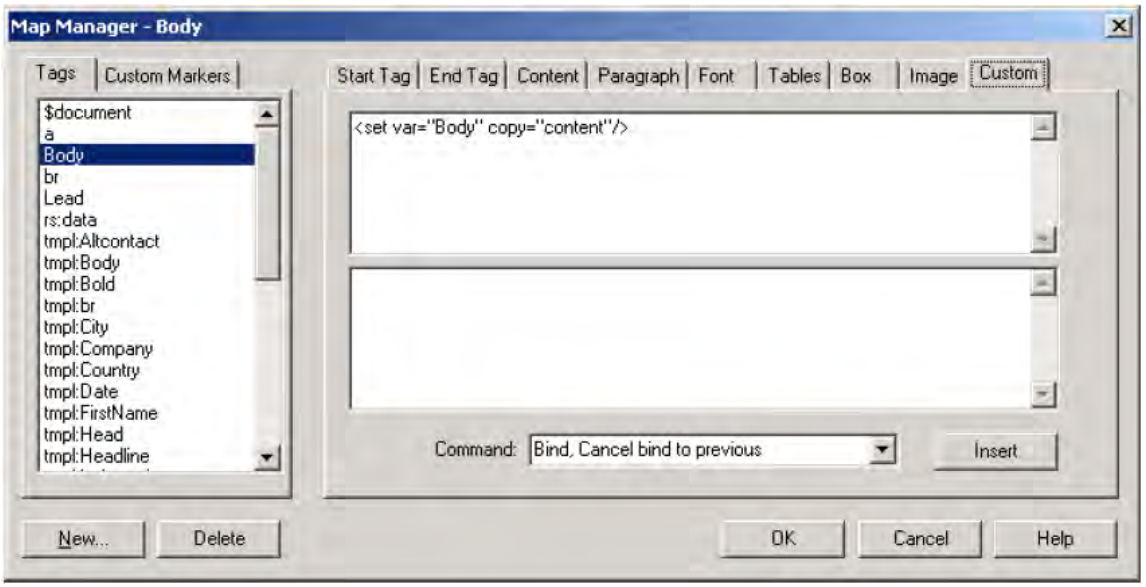
The first thing to note is that we have implemented a name space in this XML ("tmpl") to permit segregation of template-oriented mark-up from the document data. This allows us to map like-named elements differently, and keeps the whole thing nicely abstracted so that conceptually we don't get confused some later time down the road when we're trying to maintain the application.

In this template, notice that "tmpl:Headline", "tmpl:Date", "tmpl:Lead", "tmpl:Body", and "tmpl:Altcontact" are all empty tags. We've purposely named them to match the field names in the incoming data and the ASP application, again strictly for maintainability reasons. There is no intrinsic requirement to do this, however, because, to TopLeaf, a tag is just a tag.

As you can see, the template document has additional information. First, it has an overall structure that is more document-like, with headings, paragraphs, and so on, and also, it has "canned" content, in the form of contact information, banner text, and the like. For a news release, this is about all you would expect to be the same from one news item to the next. By extension, you can see that this technique could allow you to build any document you might require, and bring in customizing data when the document is composed.

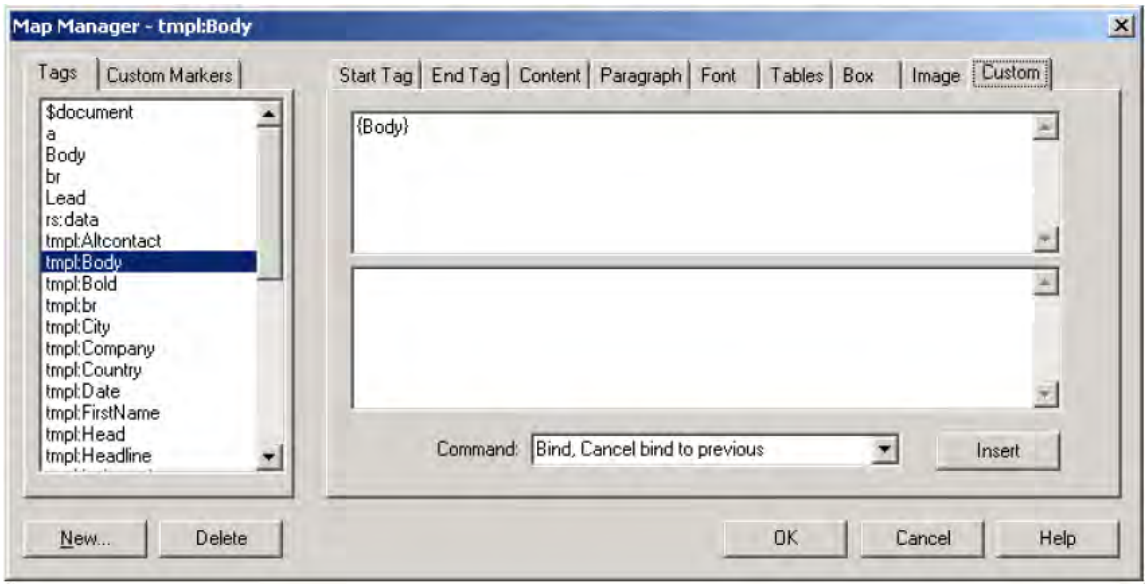
The partitions are built with the HTML table model, by the way, because they are so straightforward. The HTML table tags are not prefixed by "tmpl:", because they are not part of the template name space.

Now let's take a look at a couple of mappings.



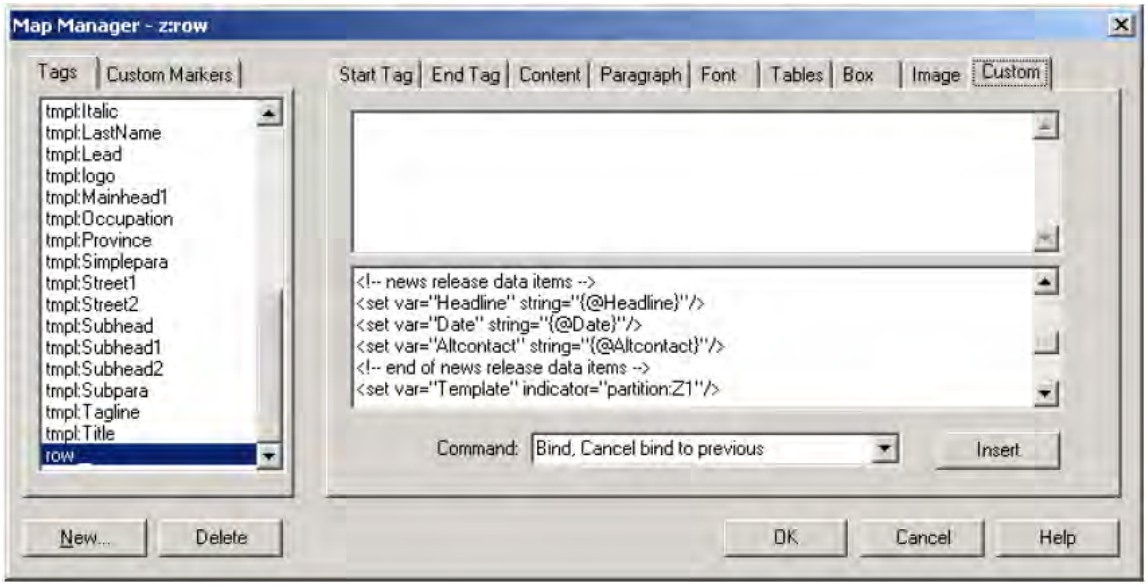
The mapping for "Body" petains to the tag in the data coming from the web form into TopLeaf. This is the "real" document as far as TopLeaf is concerned, not the template. The settings for this mapping (and all the other mappings pertaining to the web form data document) have "scan content" and "suppress content" selected. The custom box sets the variable {Body} to the content of the <Body> tag. All tags in the data document are mapped to variables this way.

The template tags (e.g., "<tmpl:Body>") emit the variable:



And that is pretty much all there is to merging data. Well, pretty much, but not completely. Where does the template come from?

The mapping for "row" is more complicated. It is the trigger for reading in the template.



Here is the custom box content for the "row" element:

```
<!-- news release data items -->
<set var="Headline" string="{@Headline}"/>
<set var="Date" string="{@Date}"/>
<set var="Altcontact" string="{@Altcontact}"/>
<!-- end of news release data items -->
<set var="Template" indicator="partition:Z1"/>
<if var="Template" target="News">
<read file="d:/topleaf/xmlcomposer/dynamic/newstmpl.xml"/>
</if>
<if var="Template" target="News" test="not-same">
```

```
<read file="d:/topleaf/xmlcomposer/dynamic/xx.xml"/>
</if>
```

First, the custom programming is assigned to the end-tag. That is, it does not execute until the end of the row is encountered. This gives the various children of the row a chance to bind to variables before this code executes.

Next, the attributes assigned to the row are bound to variables, just as the content for "Body" and "Lead" are.

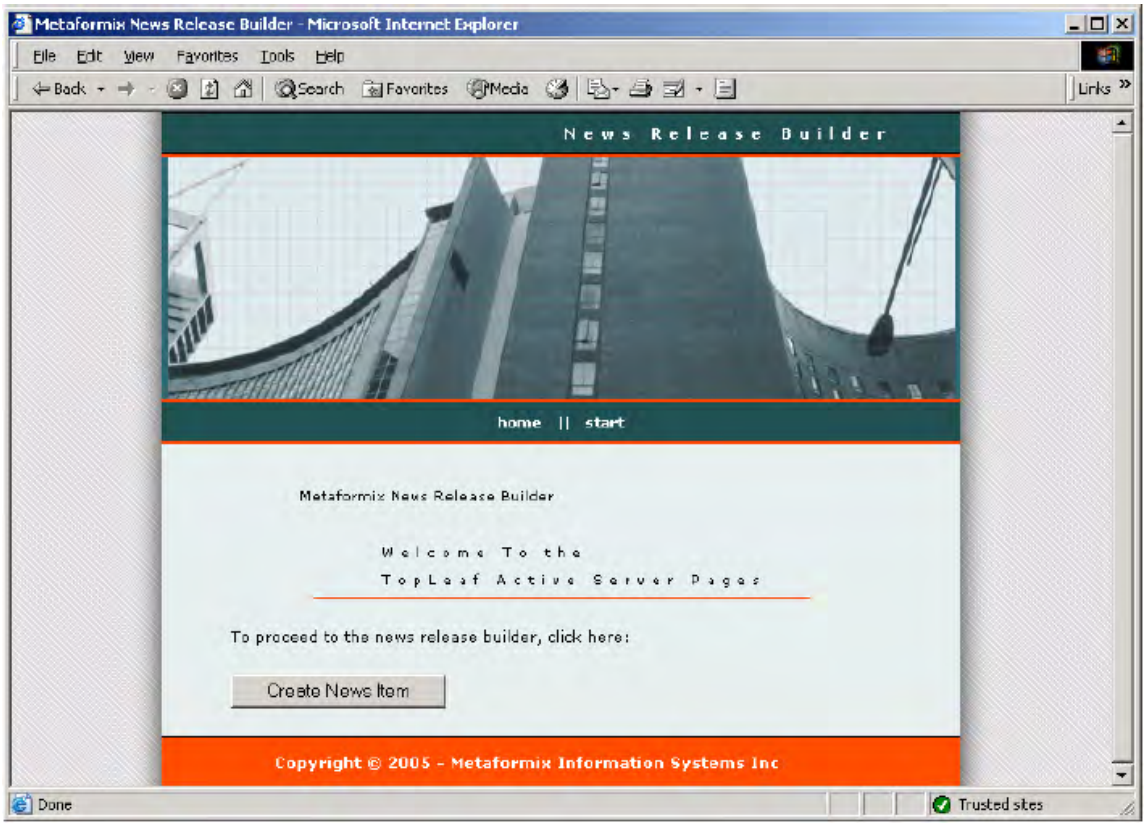
Finally, the "Z" indicator (first slot) is tested to determine which template document to use. By doing this, we can use the same publication for different purposes. Based on the result of this test, a different template is read in. Depending on the setting of the "Z" indicator, the resulting document will have the same style, but different "canned" content. If the source data is different, that is, if it came from a different application, then we would need to add mappings for those data elements into the publication.

When a "<read...>" command is executed in TopLeaf, the content of that file is read as if it existed as part of the main document's data stream. So by executing a "read" in the end-tag mapping for the data row, the tag mappings for the template are resolved. This is one way of implementing subdocument fragments in TopLeaf, too.

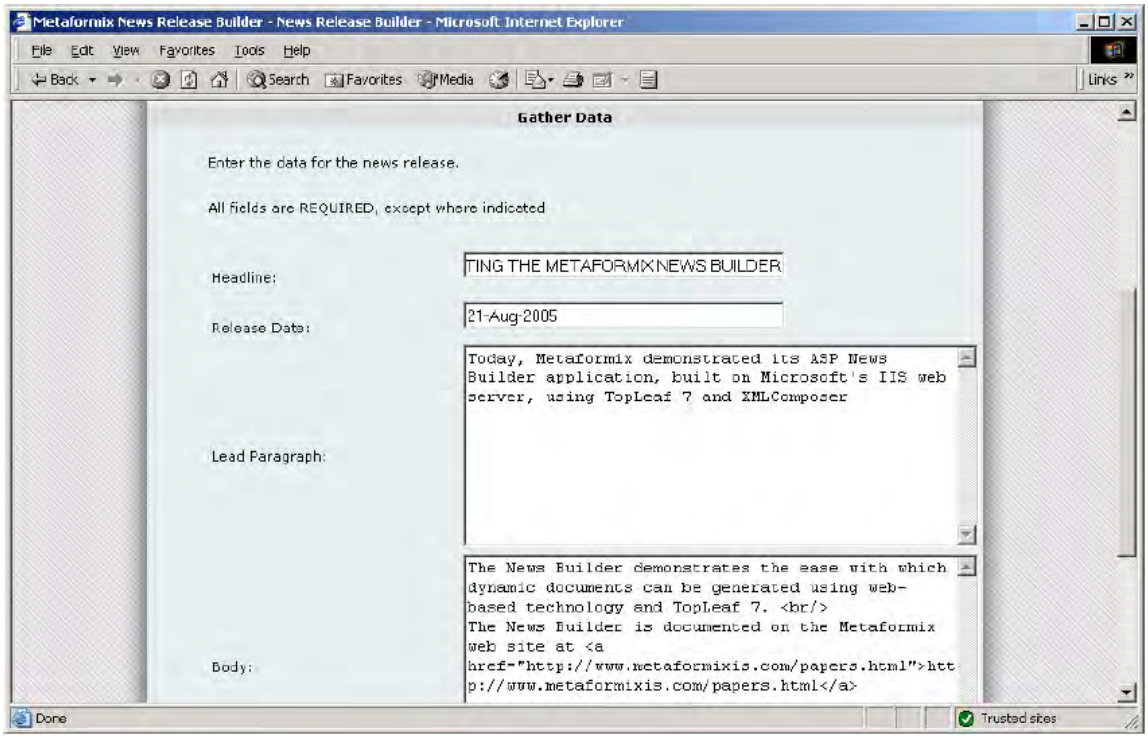
Seeing the News Builder in Action

Here are some screen captures of the application in action.

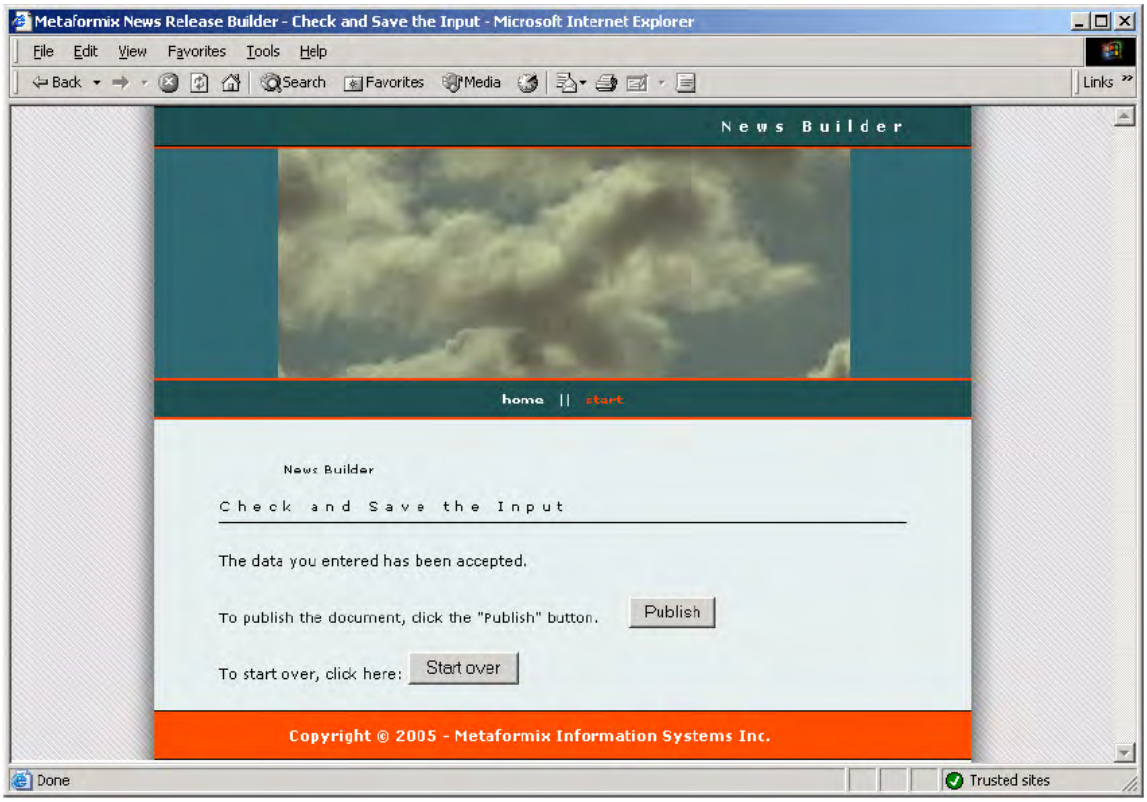
Step 1: Welcome the User (default.asp)



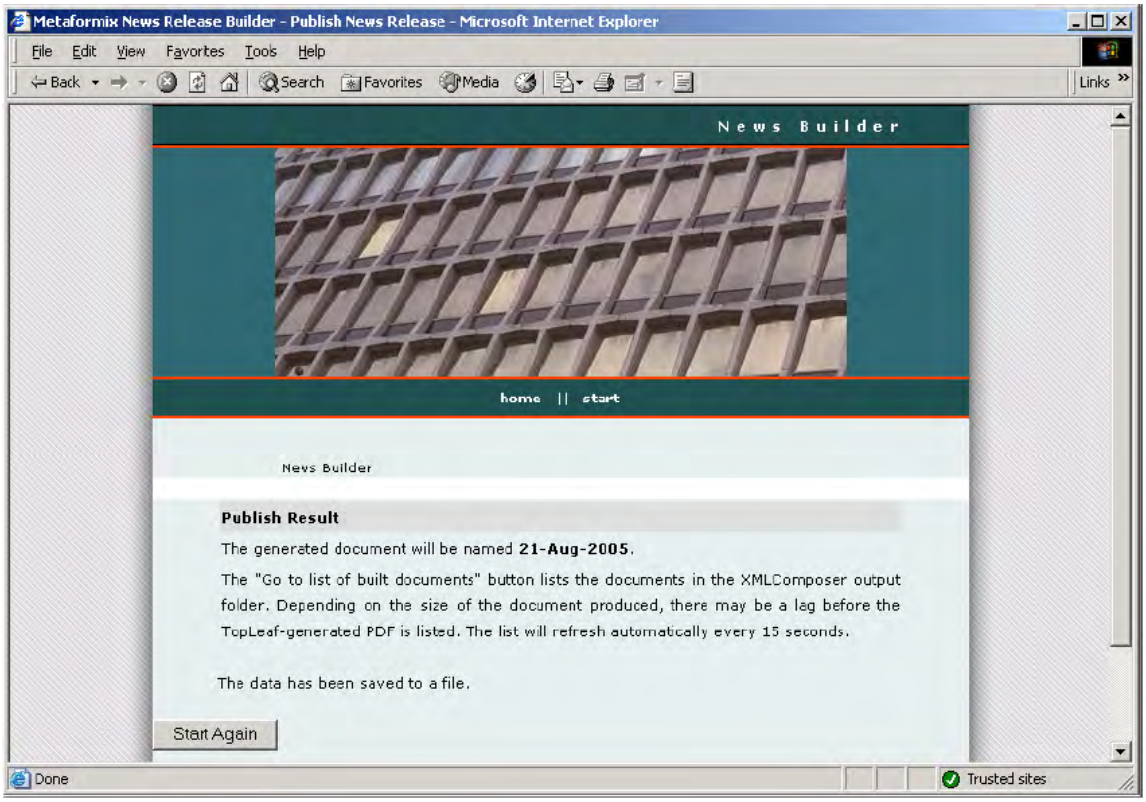
Step 2: Gather Input (MxNewsInput.asp)



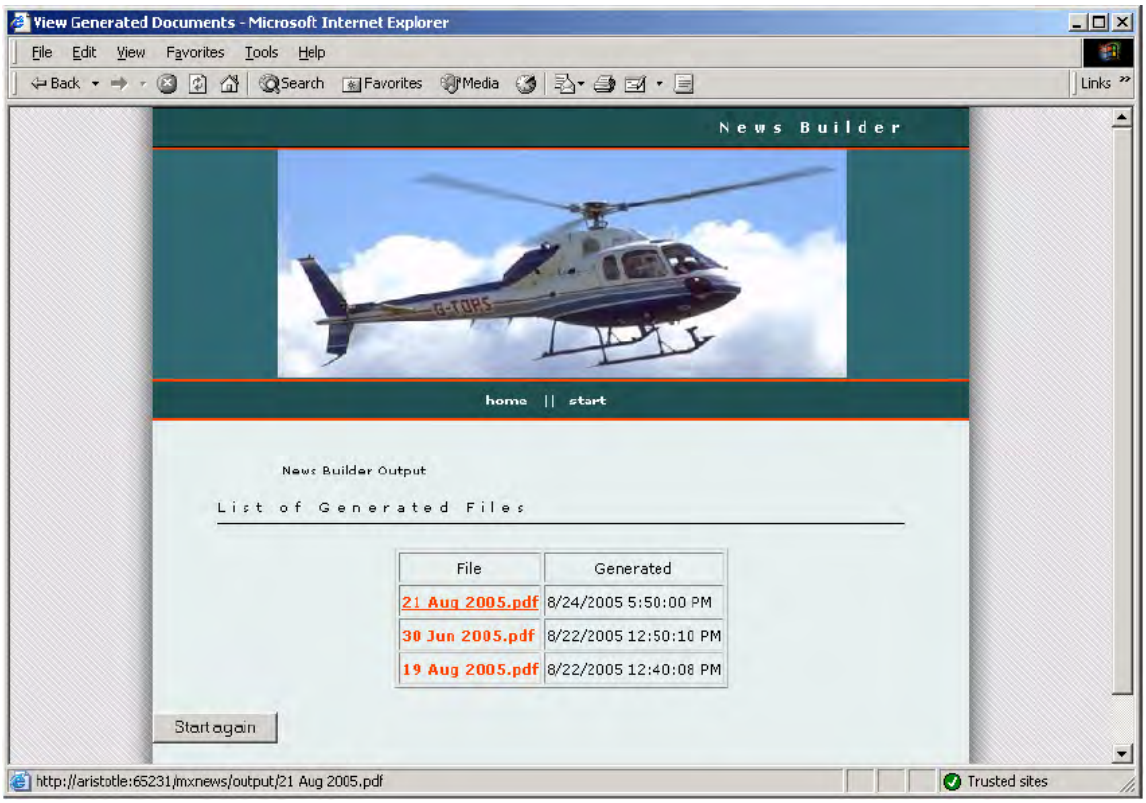
Step 3: Confirm Publication (MxNewsValidate.asp)



Step 4: Publish (MxNewsPublish.asp)



Step 5: See the Result (ViewOutput.asp)



The Composed Result

3. Conclusion

TopLeaf is a very flexible XML composition engine. In fact, it is so flexible that is is equally suited to composing everything from targetted marketing brochures, to regulatory documents thousands of pages in length. Traditionally, it has more often been used in the latter, but TopLeaf 7's custom marker features add a whole new layer of simplicity to expanding its application into publication areas normally served by dynamic or variable data publishing engines. And, at a tenth the cost of some of them, it makes template-based, variable data publishing applications accessible to nearly every company.

We've looked at how simple it is to build an ASP application by interfacing ASP with TopLeaf through XMLComposer. XMLComposer takes the grunt-work out of API programming, by providing simple instructions, and building in error checking and exception handling.

For a simple application like a news release, making an ASP application might be considered overkill. But, we hope we've demonstrated the potential of using server-based multi-tier applications with TopLeaf as the back-end composition engine to provide dynamic, on-demand publishing.

About the Author

John Barker is the co-founder and president of Metaformix Information Systems Inc. He has developed integrated publishing solutions around TopLeaf since 1998.

Contact him at john[_at_]metaformixis.com