Examining the InDesign Server Solution

This is an online bonus article for Chapter 13 of *Paperless: Real-World Solutions with Adobe Technology*. This article details the tools and techniques that were used to create the demo you read about in Chapter 13. InDesign Server provides powerful and precise graphic layout, color, and typographic automation tools for your paperless solution.

The InDesign Server Instance

The demo is running an instance of InDesign Server CS4 on a Windows 2003 Server. This instance was started by entering the following command at the Windows command prompt (**Figure 1**) after navigating to the proper directory:

```
Indesignserver -port 5501
```

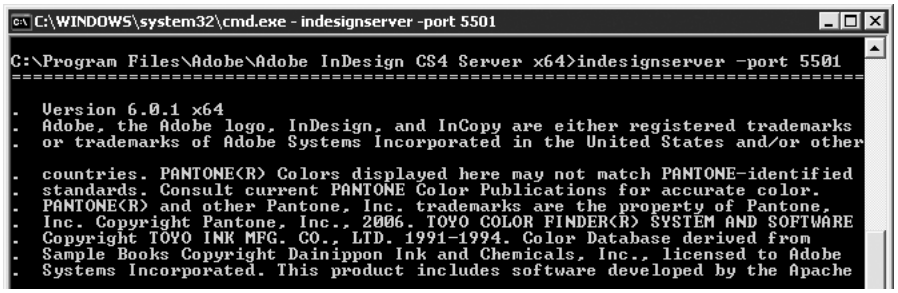

Figure 1 Your console looks like this when InDesign Server starts.

This command launches indesignserver.com, which sets up paths and launches indesignserver.exe. The command includes a port number to enable SOAP communication between the web application and the InDesign Server instance. The port can be any port that is not in use and is accessible through the firewall. When the startup sequence is complete, the "Server Running" message appears (**Figure 2**).

Figure 2 The Server Running message indicates that the server is ready to process requests.

InDesign Server can be installed with a Windows service that monitors instances of InDesign Server and reboots them when the machine is rebooted. See the book's Appendix for more information about running instances of InDesign Server.

The Master InDesign Template

The master template in the demo defines the background graphics, layout areas, and styles that will be used to generate the PDF (**Figure 3**).

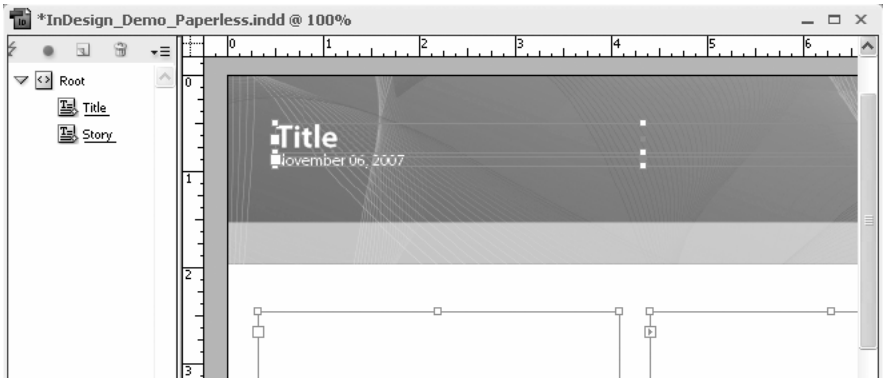

Figure 3 The template from this demo with the XML structure pane on the left and the frames highlighted on the right.

You can use the full capabilities of the InDesign desktop program to create your InDesign master template for the server. This is an advantage over other non-Adobe dynamic publishing systems that work with InDesign files. These third-party systems translate an InDesign source file into a format that will work with their systems. InDesign Server uses the native files without making any type of file transformation.

The following is a list of important elements on this template:

- **Frames:** All the content in an InDesign file is contained within frames. Figure 3 shows the frames for the title, date, and text. The text frames are linked so the XML content will flow from one frame to the other.
- **XML structure:** The template has a two-level structure that matches the top two levels of the XML files. The Root element is the topmost element and contains two child elements. The Title element is linked with the Title text frame, and the Story element is linked with the empty text frames. You can link frames with elements through a process called tagging.
- **Styles:** The template has paragraph, character, table, and cell styles that control every aspect of the finished layout. These styles can be called at runtime from a script, or they can be referenced in the XML file.
- **Text variable:** InDesign has seven types of text variables, including the Output Date variable. You can create text variables and control their behavior at runtime in the Text Variables dialog box (**Figure 4**). You can access this dialog box by choosing Type > Text Variables > Define.

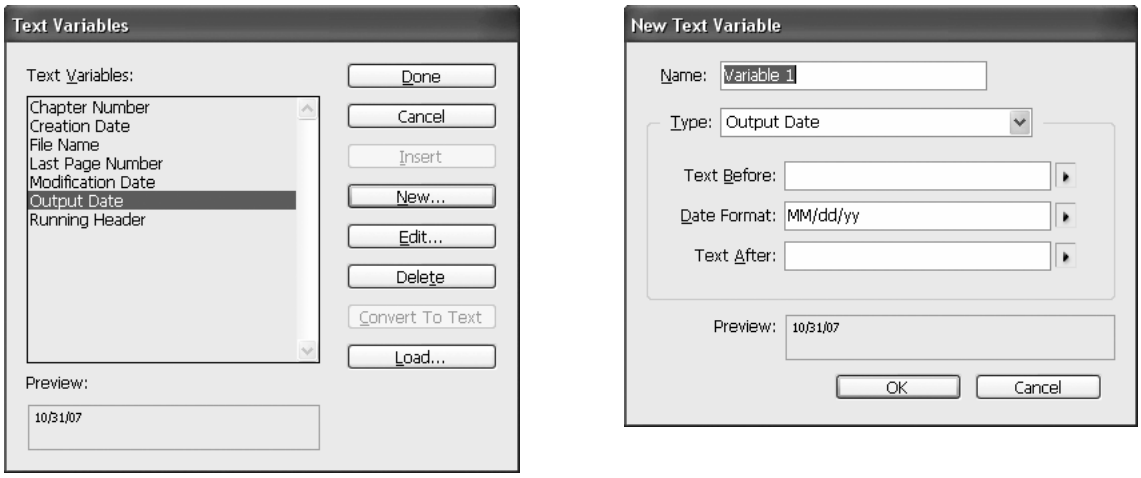

Figure 4 The Text Variables dialog box (left) and the New Text Variable dialog box (right) can be used to create text variables to add dynamic content to your master templates.

Note

The master InDesign template file used in the demo is in the standard InDesign document file format (.indd). InDesign also supports *.indt, *.inx, and *.idml file formats. INDT is a template format similar to Microsoft Word template files. INX is an XML-based format that can be read by previous versions of InDesign. IDML is new in InDesign CS4. When you save an InDesign document as IDML, InDesign creates multiple XML files that describe the InDesign document and are contained in a ZIP archive with the *.idml extension. You can edit these XML files outside of InDesign and then render them in InDesign Server.*

The XML File

The XML data files in the online demo contain the content that flows into the text frames and the styles used to format the content. The style names in the XML match the style names in the master template file. For example, the following line formats the Investment Objective string according to the Header paragraph style:

```
<Header aid:pstyle="Header">Investment Objective</Header>
```

The following paragraph of text is formatted with the Body paragraph style:

```
<Body aid:pstyle="Body">The S&amp;P 500 is a market free float-weighted index
published since 1957 of the prices of 500 large-cap common stocks actively
traded in the United States. The stocks included in the S&amp;P 500 are those
of large publicly held companies that trade on either of the two largest
American stock market companies; the NYSE Euronext and the NASDAQ OMX. After
the Dow Jones Industrial Average, the S&amp;P 500 is the most widely followed
index of large-cap American stocks. It is considered a bellwether for the
American economy, and is included in the Index of Leading Indicators. Some
mutual funds, exchange traded funds, and other funds such as pension funds, are
designed to track the performance of the S&amp;P 500 index.Thanks to the
computer technology emerging at the time, this index could be calculated and
disseminated in real time.</Body>
```

The `aid:pstyle` attribute that precedes the style name is defined in the Adobe InDesign (AID) namespace. This namespace is declared at the beginning of each XML file:

```
<Root xmlns:aid="http://ns.adobe.com/AdobeInDesign/4.0/"
xmlns:aid5="http://ns.adobe.com/AdobeInDesign/5.0/">
```

The AID namespace is also used to automatically create InDesign tables from XML content. The following XML snippet shows the first two rows of the table in the sample demo:

```
<Table aid5:tablestyle="Table" aid:table="table" aid:trows="11"
aid:tcols="3"><Cell aid5:cellstyle="TableHeader" aid:table="cell"
aid:theader="" aid:crows="1" aid:ccols="1" aid:ccolwidth="104.55">Total
Returns</Cell><Cell aid5:cellstyle="TableHeader" aid:table="cell"
aid:theader="" aid:crows="1" aid:ccols="1" aid:ccolwidth="84">S&amp;P 500□Index
Strategy</Cell><Cell aid5:cellstyle="TableHeader" aid:table="cell"
aid:theader="" aid:crows="1" aid:ccols="1" aid:ccolwidth="74.6">S&amp;P 500
Index*</Cell><Cell aid5:cellstyle="TableBody" aid:table="cell" aid:crows="1"
aid:ccols="1" aid:ccolwidth="104.55">Q4 2005</Cell><Cell
aid5:cellstyle="TableNumber" aid:table="cell" aid:crows="1" aid:ccols="1"
aid:ccolwidth="84">2.08%</Cell><Cell aid5:cellstyle="TableNumber"
aid:table="cell" aid:crows="1" aid:ccols="1" aid:ccolwidth="74.6">2.09%</Cell>
```

You can use XML with InDesign Server in many other ways as well. See the book's Appendix for more information.

The Script File

The script file in the demo provides direction to InDesign Server. It was developed in the Adobe ExtendScript Toolkit CS4 (**Figure 5**) using Adobe's ExtendScript, which is an extension of JavaScript. ExtendScript files have a *.jsx file extension.

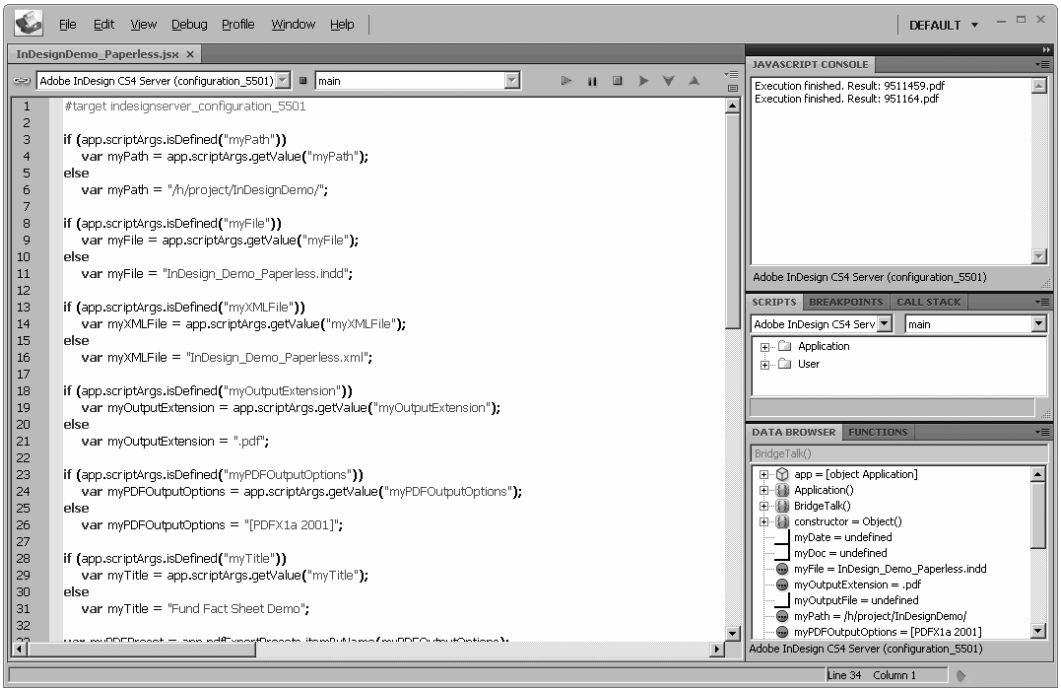

Figure 5 Adobe ExtendScript Toolkit can be run on your local machine or on your InDesign Server machine.

When you write scripts for InDesign Server, you use the InDesign Server DOM (Document Object Model). The desktop version of InDesign also has a DOM. These two DOMs are the same except for the user interface–related items that are not

relevant for the server version. For example, the demo script includes this statement to export the PDF file:

```
myDoc.exportFile(ExportFormat.pdfType, myOutputFile, myPDFPreset);
```

If this script was running on the desktop version, it would include an additional parameter to either show or hide the Export PDF dialog box. In this version of the same script, the true parameter is added to show the dialog box in the desktop version of InDesign:

```
myDoc.exportFile(ExportFormat.pdfType, myOutputFile, true, myPDFPreset);
```

The ExtendScript Toolkit includes the Object Model Viewer (OMV), which you can access by choosing Help > Object Model Viewer (**Figure 6**).

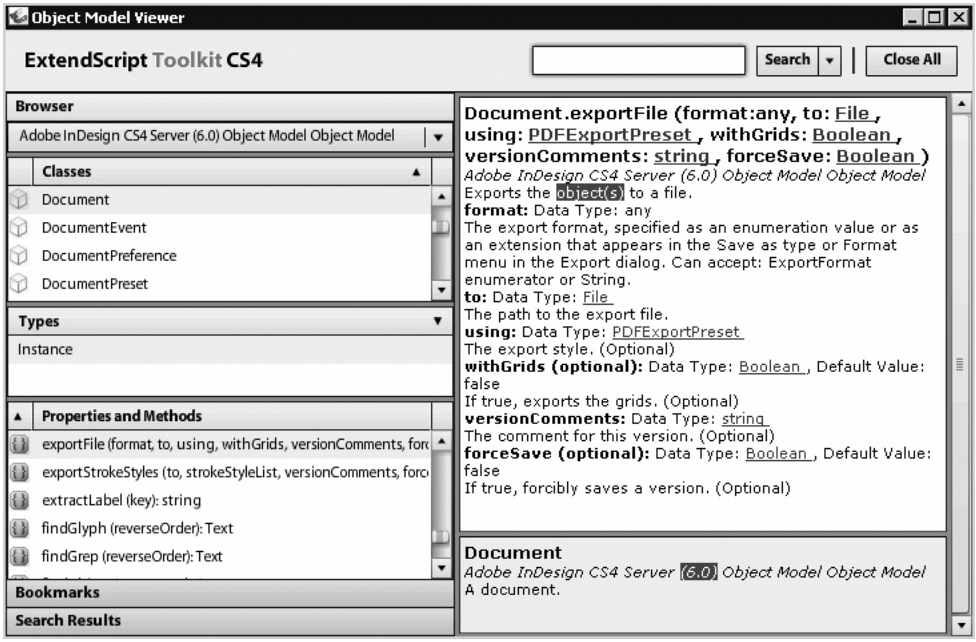

Figure 6 The Object Model Viewer in Adobe's ExtendScript Toolkit.

The Object Model Viewer enables you to view all the objects, properties, and methods of the DOM. For example, the exportFile method of the Document object is selected in Figure 6. The viewer provides a complete definition of this method on the right.

The ExtendScript Toolkit also has many other useful features for developing, testing, and debugging scripts. For example, if you want to test a script in InDesign Server, you can run the script in the ExtendScript Toolkit on the server.

The script file in the demo updates the contents of the master template and exports the file as a PDF. The following are the main actions the script takes at runtime:

1. It retrieves the scripting arguments from the SOAP call.
2. It merges the XML file with the text frames.
3. It updates the title with the user-entered data.
4. It exports the file as a PDF.

The script starts by retrieving the data from the SOAP request. The following snippet retrieves the title that the user entered and stores it in a JavaScript variable:

```
if (app.scriptArgs.isDefined("myTitle"))
        var myTitle = app.scriptArgs.getValue("myTitle");
else
        var myTitle = "Fund Fact Sheet Demo";
```

The second step merges the XML data with the text frame, which is accomplished with this line of script:

```
myDoc.importXML(File(myPath+myXMLFile));
```

The third step updates the Title text frame of the master template. Since the text frame was tagged, it can be accessed from the script with an XPath expression. ExtendScript does not natively support XPath expressions. However, Adobe provides a set of special XML processing functions in a script file named glue_code.jsx. You can reference glue_code.jsx in your script and call the processing functions to use XPath expressions to update XML content in your template. The following snippet shows how this is done in the demo file:

```
// Call glue_code.jsx
app.doScript(File("/h/project/InDesignDemo/glue_code.jsx"));

updateTitle();

function updateTitle(){
        var myRuleSet = new Array (new updateCurrentTitle);
        with(myDoc){
        var elements = xmlElements;
        __processRuleSet(elements.item(0), myRuleSet);
        }
}
function updateCurrentTitle(){
        this.name="updateCurrentTitle";
        this.xpath = "//Title";
        this.apply = function(myElement, myRuleProcessor){
        myElement.contents=myTitle;
        }
}
```

The fourth and final step exports the PDF with the following code:

```
myDoc.exportFile(ExportFormat.pdfType, myOutputFile, myPDFPreset);
```

The `myPDFPreset` parameter is a reference to a PDF preset that InDesign Server will use. This preset was selected by the user in the web page.

The SOAP Request

As you read earlier, the web client application communicates with InDesign Server through a SOAP request and response. This is the standard way to communicate with InDesign Server from a web application. An instance of InDesign Server running on an HTTP-accessible server with a unique port number exposes a web service with a method called runScript. This web service is called in the demo with the following SOAP request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap="http://ns.adobe.com/InDesign/soap/">
<soapenv:Header/>
<soapenv:Body>
      <soap:RunScript>
      <runScriptParameters>
```

```
      <scriptText></scriptText>
      <scriptLanguage>javascript</scriptLanguage>
      <scriptFile>h:\project\InDesignDemo\InDesignDemo_Paperless.jsx
      </scriptFile>
      <scriptArgs>
            <name>myXMLFile</name>
            <value>InDesign_Demo_Paperless_Line.xml</value>
      </scriptArgs>
      <scriptArgs>
            <name>myPDFOutputOptions</name>
            <value>[PDFX1a 2001]</value>
      </scriptArgs>
      <scriptArgs>
            <name>myTitle</name>
            <value>This is my Title</value>
      </scriptArgs>
      </runScriptParameters>
      </soap:RunScript>
</soapenv:Body>
</soapenv:Envelope>
```

The SOAP call to the runScript method must either include scripting code or a reference to a script file. The demo file includes a reference to a file:

```
<scriptFile>h:\project\InDesignDemo\InDesignDemo_Paperless.jsx</scriptFile>
```

The SOAP request also passes scripting arguments to runScript, including the argument that stipulates which PDF/X preset to use when rendering the file:

```
<scriptArgs>
       <name>myPDFOutputOptions</name>
       <value>[PDFX1a 2001]</value>
</scriptArgs>
```

The second part of the request/response communication mechanism is the SOAP response. InDesign Server sends back a SOAP response indicating the name of the rendered PDF file:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:IDSP="http://ns.adobe.com/InDesign/soap/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
      <IDSP:RunScriptResponse>
      <errorNumber>0</errorNumber>
      <scriptResult>
            <data xsi:type="xsd:string">95125626.pdf</data>
      </scriptResult>
      </IDSP:RunScriptResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The web application loads the completed PDF file (**Figure 7**) into the PDF container of the web page.

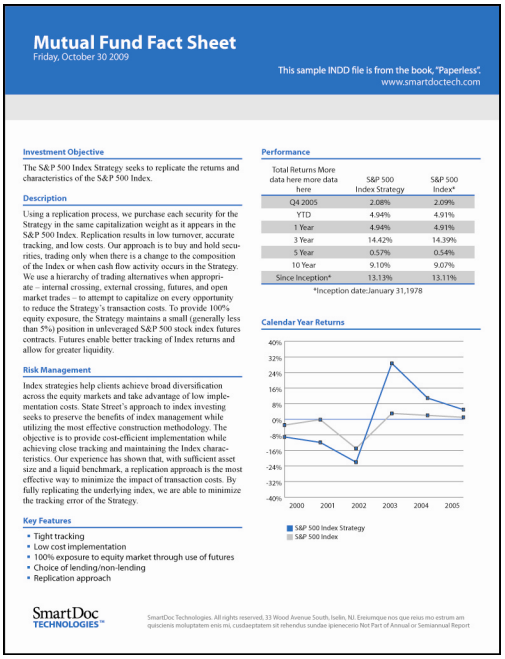

Figure 7 The finished PDF demo file with the line chart option.

There are other methods for working with InDesign Server at runtime. In addition to SOAP, you can also communicate with InDesign Server through the Java API. InDesign Server also includes a plug-in architecture that enables you to write custom plug-ins in C++. See the book's Appendix for more information on scripting and plug-in development.

Back to the Book

Now that you understand the details of an InDesign Server solution, be sure to read Chapter 14 to see how InDesign Server is used in the real world. Chapter 14 will introduce you to a financial services firm that delivers graphically rich mutual fund reports and a real estate agency that has automated its advertising production.