WEtap TopLayer

Premise

This document provides an overview of how the WEtap TopLayer product works, so you can use it to skin 3rd party applications.

Note: This guide assumes that you are familiar with the following concepts:

1. HTML
2. CSS
3. JavaScript
4. Regular Expressions
5. XSLT

Overview

The WEtap TopLayer is a tool that can modify the HTML of 3rd party applications on-the-fly. It allows a developer to change the HTML in many ways; usually with the goal of making a 3rd party application fit within an existing web site. A WEtap TopLayer installation is made up of 4 files:

1. Binary:
   /bin/WEtap.Tools.Proxyizer.dll
2. License:
   /bin/License.xml
3. Web Config:
   /web.config
4. WEtap TopLayer Config:
   /Data/Configuration.xml

Most of your efforts will be within the WEtap TopLayer Config. This configuration file is made up of two major areas:

1. Skins
   Skins provide the markup that an App will be injected into.
2. Apps
   Apps are 3rd party web sites/urls that should be injected into a skin.

Installation

Following are the simple steps needed to install the WEtap TopLayer product in IIS.

Prerequisites

Be sure you have the following installed and properly configured before installing the WEtap TopLayer product.

- IIS 7.5 or newer.
- .NET Framework v4.5.

Setup

1. File System
    a. Create a directory on your computer to run the WEtap TopLayer product from. We will refer to this directory as our Root directory.
    Example: d:\inetpub\sites\WEtap TopLayer product
    b. Within the Root directory, create a \bin subdirectory.
    c. Add the following files to your \bin subdirectory:
        i. /bin/WEtap.Tools.Proxyizer.dll
    d. Within the Root directory, create a \data subdirectory.
    e. Add the following files to your \data subdirectory.
        i. Configuration.xml
    f. Add the web.config to your Root Directory.
2. IIS
    a. Open IIS.
    b. Within your default site, add a Virtual Directory. Be sure this virtual directory's name corresponds to the virtual paths in your Configuration.xml file.
    c. Point your Virtual Directory to the Root directory created earlier.
    d. Be sure the Application Pool for your Virtual Directory is set to run .NET 4.5.
3. Test
    a. Using your web browser, navigate to the virtual directory you created, using one of the virtual paths from your Configuration.xml file.

Configuration

Settings

The configuration file allows you to use settings from the web.config. To use a web.config setting in your XML file, simply use the following syntax:

$[[((**SETTING-NAME**))]]

Where, **SETTING-NAME** is replaced with the key of your setting within your web.config.

Example:

1. Within web.config:
   <appSettings>
   <add key="WEtap.Tools.Proxyizer:Config.Location" value="~/Data/Configuration.xml"/>
   <add key="WEtap.Tools.Proxyizer:Config.Format" value="xml"/>
   <add key="Skin-URL" value="http://somesite.com/skinfile.html"/>
   </appSettings>

2. Within configuration.xml:
   <Skin Id="Base" Url="$[[((Skin-URL))]]">

3. Resulting configuration:
   <Skin Id="Base" Url="http://somesite.com/skinfile.html">

This feature should be used to keep environment-specific settings outside of your WEtap TopLayer product's configuration.

Skins

Skins are web pages that are essentially blank. The goal of the WEtap TopLayer product is to take content from an App, and inject it into a Skin. To do this, Skins must be broken down into Regions. Following is a breakdown of what makes up a Skin:

1. Id - This is the Unique Identifier for the Skin. Apps get mapped to Skins by ID, so it is important that this ID be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.
2. Url - This is the location of the skin file on the internet.
3. EnableCaching - If true, allows the skin to be cached. This can be useful when your skin changes infrequently. Use the setting CacheExpirationSeconds to set a timeout for the cached skin.
4. CacheExpirationSeconds - The number of seconds a skin should be cached before being re-requested. This setting is ignored if EnableCaching is not true.
   NOTE: If EnableCaching is true, and the Skin file throws an error (HTTP 50X, 40X, etc.), the last Skin requested will be re-used, even if it is expired.

5. Regions

   Regions define how a Skin's markup is organized so that the App's markup may be injected into the appropriate locations. There are two major types of Regions:

   a. Text

      Text regions identify a region simply by matching text. Text Regions have the following:

      i. Id: This is the Unique Identifier for the Region. Parts of Apps get mapped to Regions by their Id, so it is important that this ID be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.

      ii. Ignore Case: If true, the Region will of the Skin will be case-insensitive. If false, the Region will be case-sensitive.

      iii. Content: The content should be wrapped in a CData identifier. The content contains the text that will be used to identify the Region. If the content is "##main##", then the Region will consist of the portion of the Skin's markup that contains the first occurrence of "##main##".

   b. Regex

      Regex regions identify a region by matching a regular expression.

      i. Id: This is the Unique Identifier for the Region. Parts of Apps get mapped to Regions by their Id, so it is important that this ID be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.

      ii. RegexOptions: This contains a space (" ") delimited list of RegexOptions to be applied to the regular expression. For example, the value "IgnoreCase Singleline" tells the Region to search the Markup as though it were a single line, and ignore case.

      iii. Content: The content should be wrapped in a CData identifier. The content contains the regular expression that will be used to identify the Region. If the content is "<head>.*?</head>", then the Region will consist of the Skin's Head tag, and all of its contents.

Apps

Apps are the 3rd party applications that you want to push through the WEtap TopLayer product, and inject into your Skin.

1. Id - This is the Unique Identifier for the App. The ID should consist only of numbers, letters and dashes. No spaces.
2. IsCaseSensitive - This indicates whether the 3rd party application's web server is case sensitive. If true, then http://example.com/a would not be the same as http://example.com/A. If false, then those two urls would be treated as the same. Note, this setting only affects the URL up to the query string. Everything after the query string identifier ("?") is treated as case sensitive.
3. VirtualUrl - This represents the virtual URL that the 3rd party app will be mapped to. This should ALWAYS be a directory, never a specific file. Example: "/testapp"
4. AppUrl - This represents the root URL of the App. This should ALWAYS be a directory, never a specific file. Example: "http://www.3rdpartysite.com/somefolder"
5. SessionExpirationSeconds - This should be equal to the number of seconds a session expires

within the 3rd party application. If the 3rd party application's timeout is 20 minutes, this should be set to 20 * 60 = 1200.

6. EnableUrlCache - If true, the WEtap TopLayer product will cache URLs it has seen and re-use URL mappings. If false, every requested URL will have to be remapped between App and Virtual.
7. **Extracts**

   Extracts are responsible for extracting (or generating) Markup from a 3rd party application. When a request comes in on a Virtual Url, it is mapped to the App Url and the markup from the App is downloaded. Extracts then process the markup, extracting portions for later injection into the Skin.

   a. **Search**

      These are essentially "Regular Expression" searches of the 3rd party application's markup to find important content. Every Search has the following attributes:

      i. Id - This is the Unique Identifier for the Extract. Extracts get injected into Skin Regions by their Id, so it is important that this ID be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.

      ii. RegexOptions - This contains a space (" ") delimited list of RegexOptions to be applied to the search. For example, the value "IgnoreCase Singleline" tells the Extract to search the 3rd party application's Markup as though it were a single line, and ignore case.

      iii. CaptureGroups - If your Regular Expression contains Capture Groups, this space (" ") delimited field allows the Extract to only use the Capture Groups you care about.

      Example:

      1. If your Regular Expression was:
         (<form.*?>).*? (.*?) (.*?) (.*?)
      2. And your CaptureGroups was:
         1 2 3 4
      3. The resulting Extract would only include the portions of the Regular Expression in parentheses.

      iv. Content: The content should be wrapped in a CData identifier. The content contains the regular expression that will be used to identify the Extract. If the content is "<head>.*?</head>", then the Extract will consist of the 3rd part y application's Head tag, and all of its contents.

   b. **Static**

      This extract allows you to store markup/content within the configuration. Static extracts do not come from the 3rd party application's markup. Rather, these extracts come directly from the configuration file. Every Static Extract has the following attributes:

      i. Id - This is the Unique Identifier for the Extract. Extracts get injected into Skin Regions by their Id, so it is important that this ID be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.

      ii. Content: The content should be wrapped in a CData identifier. The content represents the HTML markup of the Extract. If the content is " The fox jumps over the log ", then the Extract will consist of that markup.

c. **XPath**

This Exact searches the 3rd party application's markup using Xpath, and extracts the contents at a particular location. Every XPath Extract has the following attributes:

i. Id - This is the Unique Identifier for the Extract. Extracts get injected into Skin Regions by their Id, so it is important that this ID be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.

ii. XPath - This is the XPath query used to find the markup in the 3rd party application.

iii. Mode - The possible values:

1. InnerHtml - Extracts the contents of the tag/container identified by the XPath.
2. OuterHtml - Extracts the tag/container identified by the XPath and its content.

d. **Xslt**

This is a very powerful Extract. It allows you to apply XSL Transformations to regions of the 3rd party application's markup. Every Xslt Extract has the following attributes:

i. Id - This is the Unique Identifier for the Extract. Extracts get injected into Skin Regions by their Id, so it is important that this ID be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.

ii. XPath- The Xslt Extract treats the 3rd party markup like a typical XML document and allows you to identify the "root" of the Extract via XPath.

iii. Content: The content should be wrapped in a CData identifier. The content represents an XSLT document used to transform the XML item identified by XPath.

e. **Custom**

Like XSLT, this is a very powerful Extract. It allows you to use C#.NET or VB.NET code to extract text from the 3rd party application's markup. Every Custom Extract has the following attributes:

i. Id - This is the Unique Identifier for the Extract. Extracts get injected into Skin Regions by their Id, so it is important that this ID be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.

ii. Class: This is the name of the Type of your custom Extractor.
Example: Custom.Proxyizer.Extractors.SampleExtractor, Custom.Proxyizer
NOTES:

1. Your custom Extractor class must implement the interface: WEtap.Tools.Proxyizer.Configuration.Apps.Extractions.ICustomExtractor
2. Be sure the assembly (*.dll) containing your custom Extractor has been dropped into the BIN directory for WEtap TopLayer product.
3. Your custom Extractor will have a method that accepts an XPathNavigator object from namespace System.Xml.XPath. The XPathNavigator will enable you to navigate through and access markup from the 3rd party application. This method will return a string.
4. If your custom extractor does not work as expected, be sure to check

the log for errors.

5. Your custom extractor will be instantiated only once for the lifetime of the application. Any state that you store in your custom extractor should take this into consideration.

8. Transforms

   Transforms perform simple find/replace logic as Extracts are Injected into a Skin. Transforms can be useful for removing unwanted tags, replacing graphics, etc. Every Transform has the following components:

   a. Id - This is the Unique Identifier for the Transform. Transforms are identified by this ID, so it is important that it be meaningful. Also, the ID should consist only of numbers, letters and dashes. No spaces.
   b. RegexOptions - This contains a space (" ") delimited list of RegexOptions to be applied to the Transform. For example, the value "IgnoreCase Singleline" tells the Transform to search the markup as though it were a single line, and ignore case.
   c. Find - A regular expression to be used to search the markup. Be sure to use parentheses to identify capture groups you might want to use in your "Replace".
   d. Replace - A replacement value. If you want to insert captures from the "Find" property, you can use standard Regex notation: $1, $2, $*n*.

9. Mapping

   Mappings are what map an App request to the correct Skin. Mappings are also responsible for injecting Extracts into the Skin's Regions.

   a. Id - This is the Unique Identifier for the Mapping. The ID should consist only of numbers, letters and dashes. No spaces.
   b. SkinId - This is the ID of the Skin that should be used for this mapping.
   c. PathRule

      Path rules are used to determine which mapping should be applied to a particular incoming URL. Each PathRule is made up of:

      i. Id - This is the Unique Identifier for the Path rule. The ID should consist only of numbers, letters and dashes. No spaces.
      ii. RuleType - Possible values are:
         1. MustPass - If this is the value, then the PathRule must pass for the mapping to be applied to the URL.
         2. MustFail - If this is the value, then the PathRule must fail for the mapping to be applied to the URL.
      iii. Regex - A regular expression used to match the URL. The regular expression will either match the URL or it will not.
      iv. Options - This contains a space (" ") delimited list of RegexOptions to be applied to the Path rule. For example, the value "IgnoreCase Singleline" tells the Path rule to search the incoming URL as though it were a single line, and ignore case.

   d. Injections

      i. SkinRegionId - This contains the ID of the region within the Skin being injected into.
      ii. ExtractIds - This contains a space (" ") delimited list of Extract IDs. Each Extract

will be injected, in the order listed.

iii. TransformIds - This contains a space (“ “) delimited list of Transform IDs. Each Transform will be applied to each Extract in the order listed.

iv. InjectOption - This controls where the Extracts will be Injected within the Skin’s Region. Possible values are:

1. Insert - This option inserts the Extracts at the beginning of the Region.
2. Append - This option appends the Extracts at the end of the Region.
3. Replace - This option clears out the Region, replacing its contents with the Extracts.

Tutorial

The following tutorial will walk you through the steps to start using the TopLayer tool in your local development environment. Follow these steps to get familiar with how the tool works.

1. Getting Started
    a. You will be provided with a copy of a Project Starter package.
    b. Unzip the starter.zip file to your local file system. Something like: c:\projects\top-layer
    c. Open the file "WEtap.Tools.Proxyizer.Starter.csproj" that was extracted within Visual Studio 2012.

2. Get Familiar
    a. You will notice several files in the project:
        i. Web.config
        This is a basic web configuration file used by IIS. The following has been set up in a DEBUG mode for you:
            1. appSettings
                a. WEtap.Tools.Proxyizer:Config.Location
                This is the location of the TopLayer configuration file. In the starter project, the path is relative to the application root; denoted by a leading "~".
                b. WEtap.Tools.Proxyizer:Config.Format
                This should be "xml". There is an experimental version that supports "json", but this should not be used in production.
                c. Proxyizer:Session:Id:Cookie
                This is the name of the session cookie that WEtap TopLayer product will create to associate user-state with a user.
            2. system.web/httpCookies
            The cookies are set up so that they can only be used over HTTP; meaning that JavaScript cannot access the cookies. As this is a developer starter file, the requireSSL is set to false. In production, the requireSSL attribute should ALWAYS be set to "true", unless your site does not use (or require) SSL.
            3. system.web/httpModules
            The http modules have been configured to add the ProxyizerModule.
            4. system.webServer/modules
            The modules have been configured to add the ProxyizerModule.
            5. log4net
            Log4Net is a logging utility. It is configured for debugging. Please refer to Log4Net's documentation to configure settings for your integration, preview and production environments.
        ii. /Data/Configuration.xml
        This is an empty TopLayer configuration file. This is where the majority of our

work will be conducted.

3. Create your first App.
   TopLayer is built on the concept of Skins and Apps. Apps are essentially pointers to applications on remote servers. Let's begin by creating our first app.
   a. Open the file /Data/Configuration.xml
   b. Add the following to before the </ProxyizerConfig>.
      <App Id="HelloWorld" IsCaseSensitive="true" KeepResourcesRemote="true" VirtualUrl="/HelloWorld" AppUrl="http://www.wetap.com/training-materials/toplayer/apps/" SessionExpirationSeconds="1200" EnableUrlCache="true"></App>

      Note the following:
      i. Remember that XML is case sensitive.
      ii. the **Id** of the App is "HelloWorld". This is how we will refer to the App throughout the tutorial. You could have used any name you like, so long as it has no spaces and is made up only of letters and dashes.
      iii. The **IsCaseSensitive** is set to true because the web server where the app is hosted sees uppercase URLs differently than lowercase URLs. Generally, 3rd party Apps hosted on IIS will be Case InSensitive (false), while Apps hosted in Apache will be Case Sensitive (true).
      iv. The **KeepResourcesRemote** is set to "true". This should always be set to "true". There is an undocumented, unsupported feature-set that allows this to be set to "false".
      v. The **VirtualUrl** is set to "/HelloWorld". This means that you will access the 3rd party application's remote App Urls locally beneath "/HelloWorld".
      vi. The **AppUrl** is set to "http://www.wetap.com/training-materials/toplayer/apps/". App Urls are ALWAYS directories, never files. With the following setup, the following files at the App Url will be mapped locally as follows:
         1. http://www.wetap.com/training-materials/toplayer/apps/index maps to /HelloWorld/index
         2. http://www.wetap.com/training-materials/toplayer/apps/step-1 maps to /HelloWorld/step-1
         3. http://www.wetap.com/training-materials/toplayer/apps/step-2 maps to /HelloWorld/step-2
         4. http://www.wetap.com/training-materials/toplayer/apps/step-3 maps to /HelloWorld/step-3
      vii. The **SessionExpirationSeconds** is set to 20 minutes. This setting is not important to this tutorial, as session state is not being used.
      viii. The **EnableUrlCache** setting is set to "true". The TopLayer application is responsible for mapping Virtual Urls to App Urls, and App Urls to Virtual Urls. If this setting is "true", TopLayer will keep track of Urls that have been previously

mapped and reuse them. If this setting is “false”, the previously mapped Urls will not be reused.

4. Try it out.
   Follow these steps to test your configuration in Visual Studio.
   a. In Solution Explorer, right click “WEtap.Tools.Proxyizer.Starter” and choose “Properties”.
   b. In the left hand menu, select “Web”.
   c. In the “Start Action” choose “Specific Page” and enter the following value in the text field: “HelloWorld/index”.
   d. Click the Save button and close the project file.
   e. In the Debug menu, select “Start Without Debugging”. In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
   f. You should see a page come up that says “Sample Application - Step Index”. Go ahead and click through the links. You have successfully mapped a 3rd party application to your local development environment.

5. Create your first Skin.
   So far, you have created a basic App that maps a 3rd party web application to your local development environment. However, the look of the App is very plain. To give the App a particular look, it must be mapped to a Skin. Let’s start by creating the Skin.
   a. Open the file /Data/Configuration.xml
   b. Add the following to before the </ProxyizerConfig>.

```
<Skin Id="Main"
Url="http://www.wetap.com/training-materials/toplayer/skins/sample"
EnableCaching="true" CacheExpirationSeconds="30">
  <Regions></Regions>
</Skin>
```

   Note the following:
   i. Remember that XML is case sensitive.
   ii. the **Id** of the Skin is “Main”. This is how we will refer to the Skin throughout the tutorial. You could have used any name you like, so long as it has no spaces and is made up only of letters and dashes.
   iii. The **Url** points to a skin page. Why not load the Url into your browser to see what it looks like.
   iv. The **EnableCaching** setting is set to “true”. This will allow the TopLayer application to reduce the number of times it requests the markup for the skin.
   v. The **CacheExpirationSeconds** is set to 30 seconds. This means that the skin’s markup will not be reloaded for 30 seconds.
   vi. The **Regions** tag is empty. We will be defining regions later on in the tutorial.

6. Map your Skin
   As you have probably noticed, just creating a Skin does not do anything. The Skin must be

mapped to an App. Follow these steps to map the Skin to the App.

a. Open the file /Data/Configuration.xml
b. Add the following child node within your HelloWorld App:
<Mapping Id="M01" SkinId="Main"></Mapping>

Note the following:

i. Remember that XML is case sensitive.
ii. the **Id** of the Mapping is “M01”. This is how we will refer to the Mapping throughout the tutorial. You could have used any name you like, so long as it has no spaces and is made up only of letters and dashes.
iii. The **SkinId** is “Main”. This corresponds with the Id of the Skin we created earlier.

c. Try it out

i. Save your work.
ii. In the Debug menu, select “Start Without Debugging”. In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
iii. You should notice that all of the following URLs have been mapped to the Skin(They are the same as the Skin Id =”Main” which is this page http://www.wetap.com/toplayer/tutorial/skins/sample), and now every page is exactly the same:

1. /HelloWorld/index
2. /HelloWorld/step-1
3. /HelloWorld/step-2
4. /HelloWorld/step-3

7. Add a Mapping Path Rule

In the previous steps, we created a Mapping that made all pages within the App show up using the Skin. Now, we will add a Mapping Path Rule to make sure that only steps 1, 2 and 3 are mapped to the skin.

a. Open the file /Data/Configuration.xml
b. Add the following child node within your HelloWorld App’s M01 Mapping:
<PathRule Id="R01" RuleType="MustPass" Regex=".*?/step.*?$" Options="IgnoreCase" />

Note the following:

i. Remember that XML is case sensitive.
ii. the **Id** of the Mapping is “R01”. This is how we will refer to the Path Rule throughout the tutorial. You could have used any name you like, so long as it has no spaces and is made up only of letters and dashes.
iii. The **RuleType** is “MustPass”. This means that the rule must pass for the mapping to be used.
iv. The **Regex** is “.*?/step.*?$”. This means that the Url must have the word “step” directly following a forward slash.

v. The Options are “IgnoreCase”. This means that the regular expression will be evaluated without consideration for case sensitivity.

c. Try it out
   i. Save your work.
   ii. In the Debug menu, select “Start Without Debugging”. In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
   iii. You should notice that all of the following URLs have been mapped to the Skin, and now look the same:
      1. /HelloWorld/step-1
      2. /HelloWorld/step-2
      3. /HelloWorld/step-3
   iv. You should notice that the following URL is not being mapped to the Skin:
      1. /HelloWorld/index

8. Inject Some Static Text

So far, all we have done is map App Urls to a Skin. Now, we will inject some static text. To do this, we need to make several changes to our Configuration file.

a. Open the file /Data/Configuration.xml
b. Add the following child node within your Main Skin’s <Regions> tag:

```
<Text Id="Footer" IgnoreCase="true"><![CDATA[##footer##]]></Text>
```

Note the following:

   i. Remember that XML is case sensitive.
   ii. the **Id** of the Text Region is “Footer”. This is how we will refer to this Region throughout the tutorial. You could have used any name you like, so long as it has no spaces and is made up only of letters and dashes.
   iii. The **IgnoreCase** is “true”. This means that the text specified will be searched for, without regard to case.
   iv. The contents of the Text Region contain a CDATA identifier with the text “##footer##”. This means that the first occurrence of the text “##footer##” within the Skin will be treated as a region. Regions are very powerful, allowing Markup to be inserted, cleared or appended. Be sure you always use a CDATA section with Text Regions.

c. Add the following child node within your HelloWorld App:

```
<Extracts>
  <Static Id="Hello-Message"><![CDATA[Hello World!]]></Static>
</Extracts>
```

Note the following:

   i. Remember that XML is case sensitive.
   ii. the **Id** of the Static Extract is “Hello-Message”. This is how we will refer to this Extract throughout the tutorial. You could have used any name you like, so long

as it has no spaces and is made up only of letters and dashes.

iii. The contents of the Static Extract contains a CDATA identifier with the text "Hello World!". This is the text that will be "Extracted" any time this static Extract is Injected into your App's Mapping. Be sure you always use a CDATA section with Static Extracts.

d. Add the following child node within your HelloWorld App's M01 Mapping:

```
<Injections>
  <Inject SkinRegionId="Footer" ExtractIds="Hello-Message" InjectOption="Replace" />
</Injections>
```

Note the following:

i. Remember that XML is case sensitive.

ii. the **SkinRegionId** of the Injection is "Footer". This means that the specified Extract(s) will be injected into the Main Skin's Footer Region.

iii. The "**ExtractIds**" field is set to "Hello-Message". This means that the injection will come from the Hello-Message Extract that we created earlier.

iv. The **InjectOption** is set to "Replace". This means that the text "##footer##" in the Skin will be replaced with the text "Hello World!" from the Hello-Message Static Extract. You could have also used Insert or Append. Insert would put the "Hello World!" text before the "##footer##" text, while Append would put the "Hello World!" text after that text.

e. Try it out

i. Save your work.

ii. In the Debug menu, select "Start Without Debugging". In a default installation of Visual Studio 2012, this is configured to CTRL+F5.

iii. You should notice that all of the following URLs have the Hello World! text in their footer.

1. /HelloWorld/step-1
2. /HelloWorld/step-2
3. /HelloWorld/step-3

9. Checkpoint

At this point, your Configuration.xml file should look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<ProxyizerConfig>
  <App Id="HelloWorld" IsCaseSensitive="true" KeepResourcesRemote="true"
VirtualUrl="/HelloWorld" AppUrl="http://www.wetap.com/training-materials/toplayer/apps/"
SessionExpirationSeconds="1200" EnableUrlCache="true">
    <Mapping Id="M01" SkinId="Main">
      <PathRule Id="R01" RuleType="MustPass" Regex=".*?/step.*?$" Options="IgnoreCase" />
      <Injections>
```

```
    <Inject SkinRegionId="Footer" ExtractIds="Hello-Message" InjectOption="Replace" />
   </Injections>
  </Mapping>
  <Extracts>
   <Static Id="Hello-Message"><![CDATA[Hello World!]]></Static>
  </Extracts>
 </App>
 <Skin Id="Main" Url="http://www.wetap.com/training-materials/toplayer/skins/sample"
EnableCaching="true" CacheExpirationSeconds="30">
  <Regions>
   <Text Id="Footer" IgnoreCase="true"><![CDATA[##footer##]]></Text>
  </Regions>
 </Skin>
</ProxyizerConfig>
```

10. Mapping Title and Content

    If we view the source of Steps 1, 2 or 3 within the App, we will notice that the step's Title is stored in an "h2" tag, while most of the content is stored within an "article" tag.

```
        <body>
            <h2>Sample Application - Step 1</h2>
<ul class="navigation">
    <li><b>Step 1</b></li>
    <li><a href="step-2">Step 2</a></li>
    <li><a href="step-3">Step 3</a></li>
</ul>
<br class="clearfix" />
<article>
    <p>This is introductory information on the form.<
    <p>Lorem ipsum dolor sit amet, consectetur adipis
dignissim hendrerit neque. Fusce sollicitudin eget du
diam aliquet. Nulla facilisi.</p>
</article>
```

    Follow these steps to map the Title and Content into the Skin.

    a. Open the file /Data/Configuration.xml

    b. Add the following Regions to your Main Skin's <Regions> tag:

       <Text Id="Title" IgnoreCase="true"><![CDATA[##title##]]></Text>
       <Text Id="Body" IgnoreCase="true"><![CDATA[##content##]]></Text>

    c. Add the following Extracts to your HelloWorld App's <Extracts> tag:

       <XPath Id="Title" XPath="//body/h2[1]" Mode="InnerHtml" />
       <XPath Id="Body" XPath="//body/article[1]" Mode="InnerHtml" />

       Note the following:

       i. Remember that XML is case sensitive.

       ii. We are now using an "XPath" extract. This type of Extract takes markup from the App, using the XPath query to locate that content.

iii. The **XPath** property indicates the location within the App where content should be found. Note the use of [1]. XPath uses 1-based indexes, not zero-based indexes. The [1] tells the Extract to only look at the first occurrence of an H2 and Article tag.

d. Add the following Injections to your HelloWorld App's M01 <Injections> tag:
<Inject SkinRegionId="Title" ExtractIds="Title" InjectOption="Replace" />
<Inject SkinRegionId="Body" ExtractIds="Body" InjectOption="Replace" />

Note the following:

i. Remember that XML is case sensitive.
ii. These two injections are mapping the Extracts to the Skin Regions. While the Ids of the Skin Regions match the Ids of the Extracts, any Ids would have worked. It is mere coincidence that the **SkinRegionId** property matches the **ExtractIds** property.

e. Try it out

i. Save your work.
ii. In the Debug menu, select "Start Without Debugging". In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
iii. First, be sure you are on the page:
/HelloWorld/index
iv. Click on each of steps 1, 2 and 3.
v. You should notice that the title and content are now being loaded into the skin.
vi. You should notice that step 2 overflows the available space. Later, we will manipulate the CSS of the Skin to create a scroll bar here.

11. Manipulating CSS

Sometimes you need to alter the CSS of the Skin for a special case. While you will want any CSS rules you create or manipulate to eventually be incorporated into the live skin, you can use the following process to test out various CSS adjustments.

a. Open the file /Data/Configuration.xml
b. Add the following Regions to your Main Skin's <Regions> tag:
<Text Id="HeadClose" IgnoreCase="true"><![CDATA[</head>]]></Text>

Note the following:

i. Remember that XML is case sensitive.
ii. This Text Region is intended to find the closing <Head> tag. This is because any styling information we want to create will be injected before the closing <Head> tag in the skin.

c. Add the following Static Extract to your HelloWorld App's <Extracts> tag:
<Static Id="Style">

```
<![CDATA[
  <style>
    body > article{
      overflow:auto;
    }
  </style>
  ]]>
</Static>
```

d. Add the following Injection to your HelloWorld App's M01 Mapping's <Injections> tag:
<Inject SkinRegionId="HeadClose" ExtractIds="Style" InjectOption="Insert" />

Note the following:

i. Remember that XML is case sensitive.

ii. This Injection has the **InjectOption** property set to "Insert". This means that the Styling information in the "Style" Extract will be inserted at the beginning of the "HeadClose" region within the "Main" Skin.

e. Try it out

i. Save your work.

ii. In the Debug menu, select "Start Without Debugging". In a default installation of Visual Studio 2012, this is configured to CTRL+F5.

iii. First, be sure you are on the page:
/HelloWorld/index

iv. Click on each of steps 1, 2 and 3.

v. You should notice that the main content area now has a scrollbar if the content cannot fit in the space provided.

12. Regular Expressions

So far we have looked at injecting static markup, as well as markup that was extracted from the App using XPath. Now, we will look at how to extract markup using Regular Expressions.

a. Open the file /Data/Configuration.xml

b. Add the following Regions to your Main Skin's <Regions> tag:
<Regex Id="DocumentTitle" RegexOptions="IgnoreCase Singleline"><![CDATA[<title>.*?</title>]]></Regex>

Note the following:

i. Remember that XML is case sensitive.

ii. This Regex Region has the **RegexOptions** property set to "IgnoreCase and SingleLine". This means that the Regular Expression used to identify the Region will treat the markup of the Skin as a single line of text for evaluation purposes, and will not respect the case of the markup.

iii. The contents of the Regex Region contain a CDATA identifier with the text "<title>.*?</title>". This means that the first match for the Regular Expression

within the Skin will be treated as a region. This regular expression will match all markup from the opening title tag to the closing title tag.

c. Add the following Extract to your HelloWorld App's <Extracts> tag:
<Search Id="DocumentTitle" RegexOptions="IgnoreCase Singleline" CaptureGroups="1"><![CDATA[(<title>.*?</title>)]]></Search>

Note the following:

i. Remember that XML is case sensitive.
ii. This Search Extract has the **RegexOptions** property set to "IgnoreCase and SingleLine". This means that the Regular Expression used to identify the Extracted markup will treat the markup within the App as a single line of text for evaluation purposes, and will not respect the case of the markup.
iii. The contents of the Search Extract contain a CDATA identifier with the text "(<title>.*?</title>)". This means that the first match for the Regular Expression within the App's markup will be used as the extract. This regular expression will match all markup from the opening title tag to the closing title tag. The parentheses surrounding the <title> tags is to create a Capture Group.
iv. This Search Extract has the **CaptureGroups** property set to "1". This tells the TopLayer engine to only extract the first Capture Group.

d. Add the following Extract to your HelloWorld App's M01 Mapping <Injections> tag:
<Inject SkinRegionId="DocumentTitle" ExtractIds="DocumentTitle" InjectOption="Replace" />

e. Try it out
i. Save your work.
ii. In the Debug menu, select "Start Without Debugging". In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
iii. First, be sure you are on the page:
/HelloWorld/index
iv. Click on each of steps 1, 2 and 3.
v. You should notice that the page's title (in the browser) is equal to what it was in the original App.

13. Custom Code
Sometimes, you may want to write custom C# code to process markup. In this case, we want to write some code to extract the navigation from the source App, and manipulate it somewhat using C#. Following are the steps to do this:
a. Note that the navigation within the Source App is located within a <ul> tag just beneath the <body>.

```
        <body>
                <h2>Sample Application - Step 1</h2>
<ul class="navigation">
    <li><b>Step 1</b></li>
    <li><a href="step-2">Step 2</a></li>
    <li><a href="step-3">Step 3</a></li>
</ul>
<br class="clearfix" />
<article>
```

b. We are going to create a C# class that locates this and creates a list of <p> tags for each link in the navigation.
   i. In your starter project, create a new folder named “Extractors”.
   ii. Within this folder, create a Class named “SidebarExtractor”.
   iii. Replace the contents of your newly created Class with the following code:

```
using System.Text;
using System.Web.UI;
using System.Xml.XPath;
using WEtap.Tools.Proxyizer.Configuration.Apps.Extractions;

namespace WEtap.TopLayer.Tutorial.Extractors
{
    public class SidebarExtractor : ICustomExtractor
    {
        public string ExtractText(XPathNavigator XPathNavigator)
        {
            //create a string builder to hold the results
            StringBuilder sBuilder = new StringBuilder();

            //get the list item iterator
            var oIterator = XPathNavigator.Select("//body/ul/li");
            while (oIterator.MoveNext())
            {
                //append the opening <p> tag
                sBuilder.Append("<p>");

                //use evaluator to go from Iterator to Node
                sBuilder.Append(DataBinder.Eval(oIterator.Current,
"CurrentNode.InnerHtml"));

                //append the closing <p> tag
                sBuilder.Append("</p>");
            }

            //return the modified markup
            return sBuilder.ToString();
        }
    }
}
```

c. Open the file /Data/Configuration.xml
d. Add the following Regions to your Main Skin’s <Regions> tag:

<Text Id="Sidebar" IgnoreCase="true"><![CDATA[##sidebar##]]></Text>

e. Add the following Extract to your HelloWorld App's <Extracts> tag:
<Custom Id="Navigation" Class="WEtap.TopLayer.Tutorial.Extractors.SidebarExtractor, WEtap.TopLayer.Tutorial"/>

Note the following:

i. Remember that XML is case sensitive.
ii. This Custom C# Extract has the **Class** property set to "WEtap.TopLayer.Tutorial.Extractors.SidebarExtractor, WEtap.TopLayer.Tutorial". This is the fully qualified type and assembly for the C# Class that we created earlier.

f. Add the following Extract to your HelloWorld App's M01 Mapping <Injections> tag:
<Inject SkinRegionId="Sidebar" ExtractIds="Navigation" InjectOption="Replace" />

g. Try it out
i. Save your work.
ii. In the Debug menu, select "Start Without Debugging". In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
iii. First, be sure you are on the page: /HelloWorld/index
iv. Click on each of steps 1, 2 and 3.
v. You should notice that the navigation in the right sidebar is now functional. If you view the page source, you will see that the navigation is using <p> tags to delimit the links instead of the <li> tags used in the original App.

14. Remapping structured markup
So far we have mapped markup Statically as well as using XPath, Regular Expressions and custom C# code. Now we will use the very powerful mechanism of XSLT to bring in the App's data entry components. When making lots of transformations to HTML markup, XSLT is very likely to be your best ally.
a. Open the file /Data/Configuration.xml
b. Add the following Extract to your HelloWorld App's <Extracts> tag:

```
<Xslt Id="DataEntry" XPath="//table">
 <![CDATA[
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output omit-xml-declaration="yes" indent="yes"/>
 <xsl:template match="table">
  <xsl:apply-templates select="node()|@*"/>
 </xsl:template>
 <xsl:template match="tr[count(td/input[@type='text']) > 0]">
  <div>
   <b><xsl:apply-templates select="td[1]"/>: </b>
   <br/><xsl:apply-templates select="td[2]"/>
```

```
  </div>
 </xsl:template>
 <xsl:template match="tr">
  <div><xsl:apply-templates select="td[2]"/></div>
 </xsl:template>
 <xsl:template match="td">
  <xsl:apply-templates select="node()|@*"/>
 </xsl:template>
 <xsl:template match="node()|@*">
  <xsl:copy>
   <xsl:apply-templates select="node()|@*"/>
  </xsl:copy>
 </xsl:template>
</xsl:stylesheet>
]]>
    </Xslt>
```

Note the following:

i. Remember that XML is case sensitive.
ii. This XSLT Extract's **XPath** property points to the root element within the App's markup that you plan to process with the XSLT.
iii. Content: The content should be wrapped in a CData identifier. The content contains the XSLT code that will be used to modify the App's Markup.

c. Add the following Extract to your HelloWorld App's M01 Mapping <Injections> tag:
<Inject SkinRegionId="Body" ExtractIds="DataEntry" InjectOption="Append" />

d. Try it out
   i. Save your work.
   ii. In the Debug menu, select "Start Without Debugging". In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
   iii. First, be sure you are on the page: /HelloWorld/index
   iv. Click on each of steps 1, 2 and 3.
   v. You should notice that on steps 1 and 2 a form is present. If you view the source, you will see that instead of the original App's <table> tags, we have tags.

15. Moving settings outside of Configuration.xml
Many times you will have environment-specific variables that you want to incorporate into your Configuration.xml. For instance, the URL to the Skin file might be different in your local Development environment than it is in Production. Follow these steps to move these values out of the Configuration.xml and into the web.config.
    a. Open the file /web.config

b. Add the following Setting to your appSettings:
   <add key="MainSkinUrl"
   value="http://www.wetap.com/training-materials/toplayer/skins/sample"/>
c. Open the file /Data/Configuration.xml
d. Modify your Main Skin so that the Url property is: $[[((MainSkinUrl))]]

e. Try it out
   i. Save your work.
   ii. In the Debug menu, select “Start Without Debugging”. In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
   iii. First, be sure you are on the page: /HelloWorld/index
   iv. Click on each of steps 1, 2 and 3. Your skin file should continue to load as before, even though it is not directly referenced in the Configuration.xml file.

16. Mapping <form> tags

One thing that this tutorial did not specifically cover is <form> tags. <Form> tags are not directly accessible via Xslt or XPath. This is because <form> tags are not addressable by these .NET parsers. To successfully get an Opening and Closing form tag, it is recommended that you use Regular Expressions to map the opening and closing <form> tags into your skin.

a. Try it out
   i. Use the logic you have learned so far in the tutorial to create a Search Extract to find the opening and closing <form> tags in the App.
   ii. Use what you have learned so far to create Injections to map the Extracted <form> tags into your skin.
   iii. Save your work.
   iv. In the Debug menu, select “Start Without Debugging”. In a default installation of Visual Studio 2012, this is configured to CTRL+F5.
   v. First, be sure you are on the page: /HelloWorld/index
   vi. Click on each of steps 1, 2 and 3. View the source of steps 1 and 2 to see if your form tag is present.