Novell
exteNd
Composer

5 . 2

HTML CONNECT USER'S GUIDE

Novell.

Novell Trademarks

ConsoleOne is a registered trademark of Novell, Inc.

eDirectory is a trademark of Novell, Inc.

GroupWise is a registered trademark of Novell, Inc.

exteNd is a trademark of Novell, Inc.

exteNd Composer is a trademark of Novell, Inc.

exteNd Director is a trademark of Novell, Inc.

iChain is a registered trademark of Novell, Inc.

jBroker is a trademark of Novell, Inc.

NetWare is a registered trademark of Novell, Inc.

Novell is a registered trademark of Novell, Inc.

Novell eGuide is a trademark of Novell, Inc.

SilverStream Trademarks

SilverStream is a registered trademark of SilverStream Software, LLC.

Third-Party Trademarks

All third-party trademarks are the property of their respective owners.

Third-Party Software Legal Notices

The Apache Software License, Version 1.1

Copyright (c) 2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

JDOM.JAR

Copyright (C) 2000-2002 Brett McLaughlin & Jason Hunter. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution. 3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jdom.org. 4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management (pm@jdom.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the JDOM Project (http://www.jdom.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.jdom.org/images/logos.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun

Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun Logo Sun, the Sun logo, Sun Microsystems, JavaBeans, Enterprise JavaBeans, JavaServer

Pages, Java Naming and Directory Interface, JDK, JDBC, Java, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultraserver, Where The Network Is Going, SunWorkShop, XView, Java WorkShop, the Java Coffee Cup logo, Visual Java, and NetBeans are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Indiana University Extreme! Lab Software License

Version 1.1.1

Copyright (c) 2002 Extreme! Lab, Indiana University. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Indiana University" and "Indiana University Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/. 5. Products derived from this software may not use "Indiana University" name nor may "Indiana University" appear in their name, without prior written permission of the Indiana University.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Phaos

This Software is derived in part from the SSLavaTM Toolkit, which is Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved. Customer is prohibited from accessing the functionality of the Phaos software.

W3C

W3C® SOFTWARE NOTICE AND LICENSE

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications: 1.The full text of this NOTICE in a location viewable to users of the redistributed or derivative work. 2.Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code. 3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Contents

About This Book

Purpose

This guide describes how to use the HTML Component Editor, which is the design-time portion of the exteNd Composer HTML Connect.

Audience

This book is for systems analysts, programmers, and others who intend to build applications or services that require a web-page "screen scraper" component capable of accessing non-secure as well as secure web pages using HTTP basic authentication, digest authentication, or NTLM security procedures.

Prerequisites

This book assumes prior familiarity with the exteNd Composer design-time environment and Composer application-building metaphors. You should also be familiar with HTML, HTTP session behavior, HTTP error messages, and related concepts.

Additional documentation

For the complete set of Novell exteNd exteNd Director documentation, see the Novell Documentation Web Site:

http://www.novell.com/documentation-index/index.jsp.

1 **Welcome to exteNd Composer and HTML Connect**

Welcome to the *Connect for HTML User's Guide*. This Guide is a companion to the *exteNd Composer User's Guide*, which details how to use all the features of Composer, except the Connect Component Editors. So if you haven't looked at the User's Guide yet, please familiarize yourself with it before using this Guide.

Novell exteNd Composer provides separate Component Editors for each Connect, like HTML. The special features of each component editor are described in separate Guides like this one.

If you have been using exteNd Composer, and are familiar with the core component editor, the XML Map Component Editor, then this Guide should get you started with the HTML Component Editor.

Before you can begin working with the HTML Connect you must have installed it into your existing exteNd Composer. Likewise, before you can run any Services built with this Connect in the Composer Enterprise Server environment, you must have already installed the Server side software for this Connect into Composer Enterprise Server.

NOTE: To be successful with this Component Editor, you must be familiar with the HTML environment and the applications that you want to XML-enable.

About exteNd Connects

Novell exteNd Composer is built upon a simple hub and spoke architecture. The hub is a robust XML transformation engine that accepts requests via XML documents, performs transformation processes on those documents (with or without the participation of other XML-enabled services), and returns an XML response document. The spokes, or *Connects*, are plug-in modules that "XML-enable" sources of data that are not XML aware, bringing their data into the hub for processing as XML. These data sources can be anything from legacy applications to Message Queues to HTML pages, as shown below.
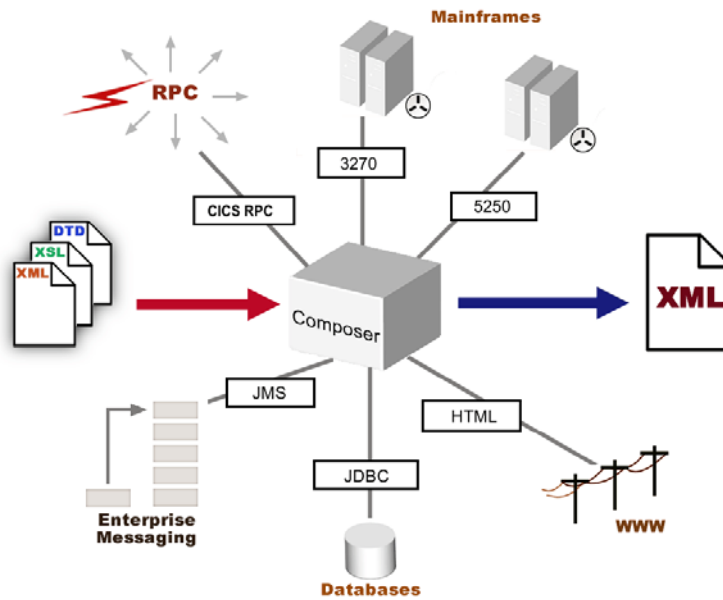
Composer Connects can be categorized by the integration strategy each one employs to XML enable an information source. The integration strategies are a reflection of the major divisions used in modern systems designs for Internet-based computing architectures. (See illustration below.) Depending on your B2B needs and the architecture of your legacy applications, Novell exteNd can integrate your business systems at the User Interface, Program Logic, or Data levels.
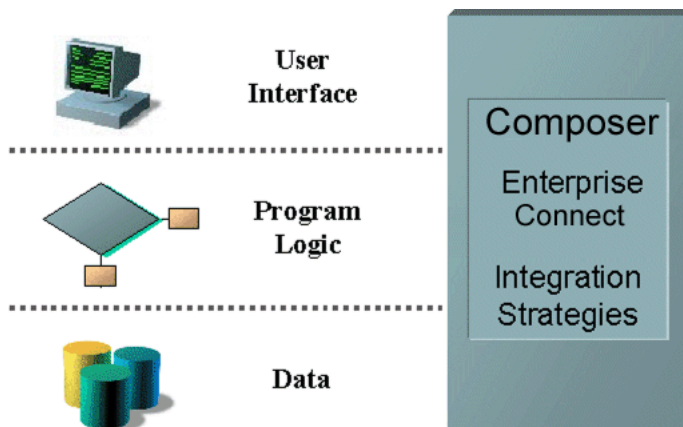
What is the HTML Connect?

Composer's HTML Connect allows you to record navigation and interactions with standard Web Sites using the User Interface integration strategy by hooking into the HTML information stream (like a browser does) for later playback as a B2B service. The term HTML (Hyper Text Markup Language) is used for documents published on the World Wide Web. The HTML Connect uses a utility called Tidy which transforms hard to read and often syntactically incorrect HTML information into a clearly layered XHTML document.

NOTE: The HTML Connect does not process Flash, applets, nor ActiveX controls. JavaScript is supported: see discussion further below.

HTML page appears in the Native Environment Pane
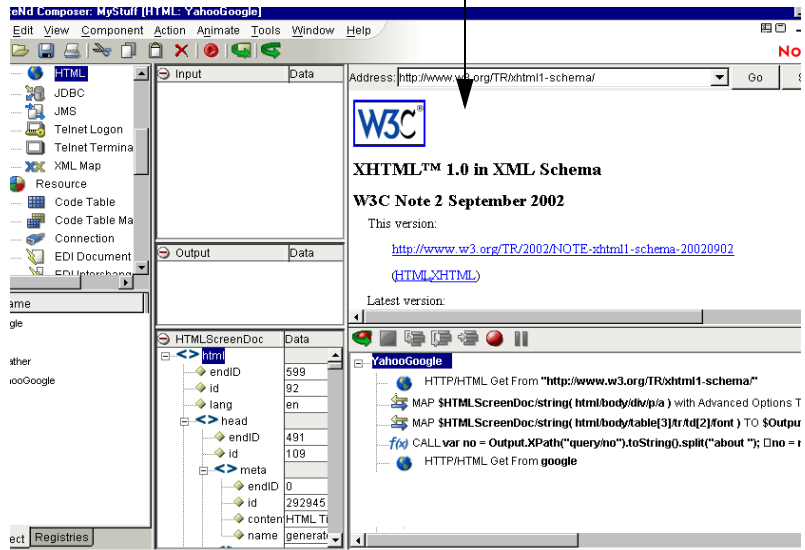
About Composer's HTML Component

Much like the XML Map component, the HTML Component is designed to map, transform, and transfer data between two different XML templates (i.e., request and response XML documents). However, it is specialized to make a connection to an Internet or Extranet application as viewed by a web page, process the data using elements from a DOM, and then map the results to an output DOM. You can then act upon the output DOM in any way that makes sense for your integration application.
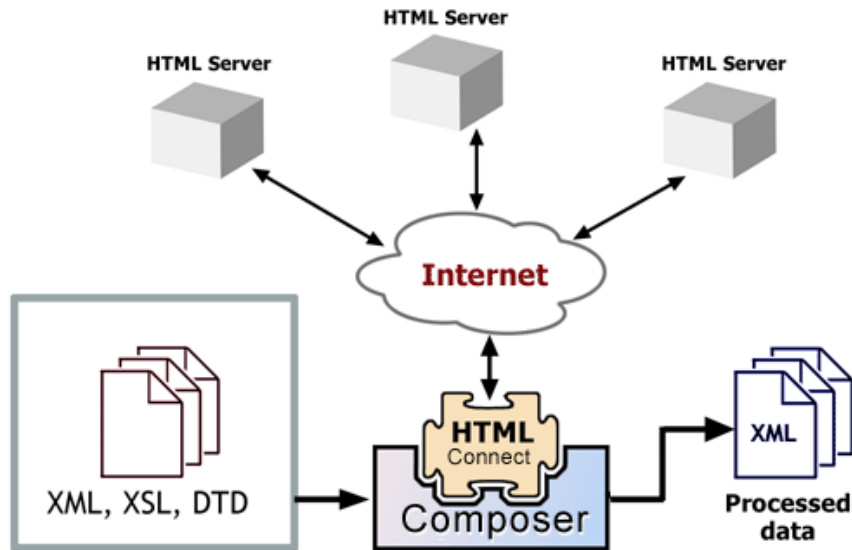
An HTML Component can perform simple data manipulations, such as mapping and transferring data from an XML document into an HTML form, or perform "screen scraping" of an HTML transaction, putting the data into an XML document. The HTML Component has all the functionality of the XML Map component and can process XSL, send mail, and post and receive XML documents using the HTTP protocol.

About Secure Socket Layer Support (SSL)

The HTML Connect is capable of providing secure connections between exteNd Composer and the target Web Site. This security is accomplished by using Secure Socket Layer Support 3 (SSL.3) Composer supports both client and server side digital certificates. The HTML Connect uses the HTTP Basic Authentication connection resource type to set up the security to interact with sites on the internet. The SSL protocol allows for authenticated and encrypted communication between the browser and server. To enable this security, the Composer application must have access to, and be able to process the digital certificates required by, the target web site. This means that these certificates must be installed on the your web server (Novell, Jakarta Tomcat, WebLogic, or Websphere) as well as in the HTML Connect. To accomplish this, Composer provides two methods. For server-side certificates, the HTML Connect ships with approximately 100 industry-standard digital certificates in the jar file *agrootca.jar,* located in the Designer/lib directory.

If you need to add a server-side certificate that is not included, refer to the documentation provided with your application server on how to maintain certificates. Also refer to the appendix for a list of certificates that are supported and how to add a new server-side certificate in Composer. For client-side certificates, Composer supports DER encoded binary x509 certificates. To associate the certificate, Composer allows you to specify the certificate in the HTTP Basic Authentication connection resource.

Other Protocols

The HTML Component Editor supports only the HTTP/S protocol. URLs that use **file://**, **ldap://**, **ftp://**, or other schemes are not honored.

NOTE: The **file://** protocol is supported by Composer's URL/File Read and URL/File Write actions (which can be used in any kind of component, not just HTML). Consult the chapter on Advanced Actions in the Composer User's Guide for more information.

About Cookies

A cookie is a text file that gets stored by the browser. One main purpose of cookies is to identify the user (the HTTP client) to the host: for instance, to maintain log-on status when moving from page to page in a website, or to maintain the current session while shopping using a shopping cart on a website. Cookies are general mechanisms which server side connections can use to both store and retrieve information on the client side of the connection. There are two types of cookies: session cookies that are active only for the duration of the session and persistent cookies that are kept on disk in the computer and are available each time you access that site.

The HTML Connect supports *session* cookies. This means that if required by a site, the HTML Connect will create a cookie for the time that the component is active. The cookie will be set once and then go away when the component finishes executing. During testing, the restart of animation or the execution of the component will discard any cookies before execution begins, as each execution of the component is treated as a new session. Cookie persistence is not supported.

About Frames

This version of the HTML Connect supports pages with frames. The *HTMLScreenDoc* will contain a DOM for a loaded page. When you click on the HTML panel or drag and drop information into the input controls of one of the frames, the HTMLScreenDoc will display a DOM for this frame. If the frame is to be changed at this time, a new SetFrame action is created and recorded in the Map Action.

About JavaScript

The HTML Connect supports JavaScript in web pages, to the extent necessary to process "redirects" and capture your design-time interactions with forms and form widgets so that you can work with scripted pages in the HTML Component Editor in seamless fashion.

There are certain limitations to JavaScript support in the HTML Component Editor. The limitations are mainly rendering-related. For example, you will not see button-rollover effects, and it's possible you will not be able to use a page that relies heavily on JavaScript-powered CSS or "DHTML" effects. In many cases, pages will *look* different in the component editor's native environmental panel than in your regular web browser, but you will still be able to "record" your interactions with the page so as to capture the kinds of data you need and/or map your own data into a webform. The only way to know for sure is to try working with a particular web page.

Another thing to be aware of is that many web-page authors choose to use nonstandard, browser-specific JavaScript extensions that a standards-strict HTML client will not be able to understand. Because there are so many possible permutations of browser version, language version, HTML and DOM versions, nonstandard language extensions, nonstandard vendor *implementations* of standards-based language extensions, etc., it is impossible to predict whether a given web page's scripts will execute properly in Composer's HTML Component Editor, without testing.

NOTE: Many web sites, especially those with mission-critical webforms, offer non-JavaScript versions of their pages for use by customers who either have older browsers or have chosen to turn JavaScript off. When this is the case, you should use the non-JavaScript version of the web site (or web page) when building your components.

What Applications Can You Build Using the HTML User Interface Component Editor?

The HTML User Interface Component Editor allows the extension of any XML integration you are building to include any of your business applications that require HTML-based interactions (See *exteNd Composer User's Guide* for more information.) For example, you may have an application that retrieves a product's description, picture, price, and inventory from regularly updated databases and displays it on a Web browser. By using the HTML Component Editor, you can now get the current product information from the website and the static information (e.g., the picture) from the database and merge the information from these separate information sources before displaying it to a user. This provides the same up-to-date information to both your internal and external users.

2 Getting Started with the HTML Component Editor

The Sample Transactions

For demonstration purposes, one transaction is used throughout this document in the sample presented. You will be navigating to a Web site and entering a SKU number into a form to drive an inquiry. Also, you will change various options to retrieve different information from the Web site screen. You will be able see the details of the HTML Web page by viewing an XHTML representation of the Web page in an object called the ScreenDoc DOM. You can see the interaction of your selections and the result of them in the Action Pane. You will also be able to modify and edit your actions before saving your service.

Steps Commonly Used to Create an HTML Component

There are many ways to go about creating HTML Components; however, the most commonly used steps in creating a simple HTML Component are as follows:

1   Create XML Templates required for any Inputs into the HTML or Outputs from the HTML transaction.

2   If you are going to access a secure site, create a Connection Resource containing any necessary security information.

3   Create an HTML Component.

4   Enter Record mode and navigate to Web page(s) for the information you want to capture.

5   Drag and drop information from the Input Part into the Web page to drive form interactions and drag and drop results from the Web page into the Output Part.

6   Edit actions if necessary.

7   Execute and test the component.

8   Save the component.

Proxy Server Settings

If your organization requires web users to go through a proxy server in order to get to the Web, you will need to provide proxy settings that Composer can use for "tunneling out" through the proxy.

NOTE: This is both a design-time and runtime issue. See further discussion below.

➢ **To configure Proxy Server Settings:**

1   In Composer's **Tools** menu, choose **Configuration** to bring up the Configuration dialog (as shown above).

2   Click the checkbox labelled **Use a proxy server**. The **Advanced** button at the far right of the dialog will become enabled, along with the text fields labelled Address and Port.

3   In the **Address** text field, enter the IP address of the proxy server.

4   In the **Port** field, enter the appropriate port number.

5   To expose additional settings, click the **Advanced** button. A new dialog appears:
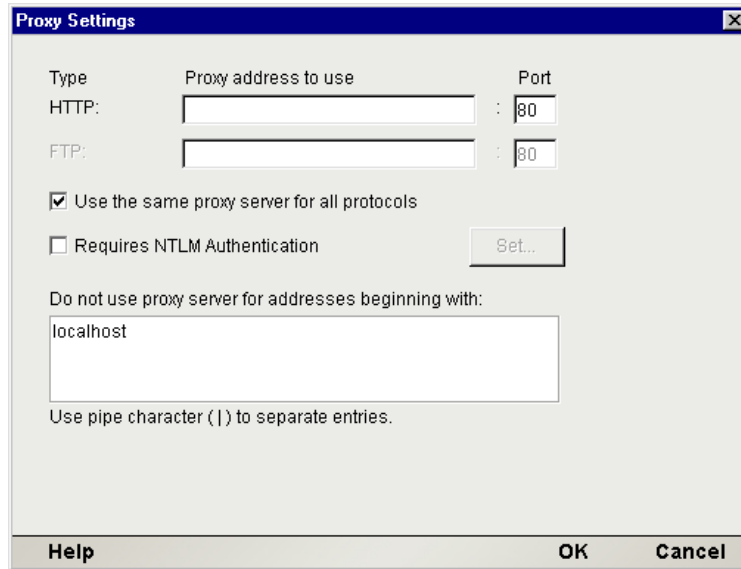
6   If you wish to specify a different proxy address and/or port for FTP access than for HTTP access, make sure the **Use the same proxy server for all protocols** checkbox is *unchecked*. (Otherwise, the FTP text field will remain disabled.) Then supply the IP address and Port information for the FTP proxy in the text fields provided.

7   If your proxy server requires NTLM authentication in order to access its services, check the **Requires NTLM Authentication** checkbox.

8   If you did not check the **Requires NTLM Authentication** checkbox, continue to step 13. The **Set** button will be enabled if you have checked the **Requires NTLM Authentication** checkbox. Click this button. A new dialog appears.
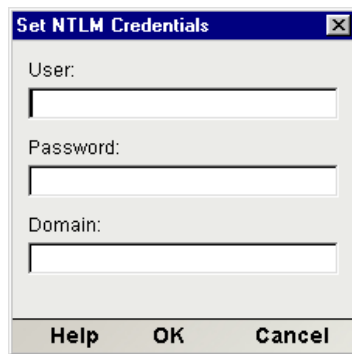
9   In the **User** field, enter the user name that you were issued for authentication.

10   In the **Password** field, enter your password.

11   In the **Domain** field, enter the name of the realm to which this authentication procedure applies.

12   Click **OK** to dismiss the dialog.

13   In the Proxy Settings dialog, in the text area labelled **Do not use proxy server for addresses beginning with**, enter a domain name if you wish to exclude certain domains from the authentication-handshake procedure. (A common case here is that you might want to exclude *localhost* and/or other in-house test domains.) You may enter multiple domains, separated by the pipe character.

14   Finally, use **OK** to dismiss the dialog.

Proxy Settings for Runtime

If your HTML Components, once deployed, will still need to "tunnel out" through a proxy, you will want to carry your proxy settings over to the runtime environment.

The proxy settings you enter in the Configuration dialog are stored in a file called **xconfig.xml** (in Composer's **\bin** directory). In order for those settings to remain active for deployed components, you will need to make sure that the **xconfig.xml** file for Composer Enterprise Server (which exists on the app server) contains your proxy info. Open the design-side **xconfig** file in a text editor and look for the PROXYSERVERINFO element. You will see a section of data that looks approximately like:

```
<PROXYSERVERINFO>
     <USEPROXYSERVER Desc="If on, the additional PROXY options are enabled (valid
values are on | off)">on</USEPROXYSERVER>
     <HTTPPROXYHOST Desc=" For Doc I/O, HTTP Actions etc., if network uses a
proxy enter name here.">127.7.7.7</HTTPPROXYHOST>
     <HTTPPROXYPORT Desc="Port number HTTPPROXYHOST listens
on.">8008</HTTPPROXYPORT>
     <HTTPNONPROXYHOSTS Desc="List of hosts that do not require a Proxy.  Each
hostname must be seperated by a pipe &apos;|&apos.">localhost</HTTPNONPROXYHOSTS>
     <FTPPROXYHOST Desc=" For Doc I/O, HTTP Actions etc., if network uses a proxy
enter name here.">127.7.7.7</FTPPROXYHOST>
     <FTPPROXYPORT Desc="Port number FTPPROXYHOST listens
on.">8008</FTPPROXYPORT>
     <NTLMCREDENTIALS>
       <PROXYNTLMPROTECTED>on</PROXYNTLMPROTECTED>
       <NTLMUSER>MikeM</NTLMUSER>
       <NTLMPWD>ABYZsjbDcOk=
           </NTLMPWD>
       <NTLMDOMAIN>Argonaut</NTLMDOMAIN>
     </NTLMCREDENTIALS>
</PROXYSERVERINFO>
```

Copy this section over to your server-side **xconfig.xml** file.

About HTML Connection Resources

The Enterprise Connect for HTML is somewhat different than other types of Connectors in that no special connection resources need to be created in order to use it, as long as there are no special security requirements for visiting sites. The only time you will need to create special connection resources for your HTML Component is when one or more of your HTML Actions will be accessing secure web pages through HTTPS, Digest HTTP authentication, or NTLM (NT LAN Manager). In that case, you will need to set up one or more HTTP connection resources (as described further below) for *each* secure site that you intend to visit. You will then assign the appropriate connection resource to each HTML Action in your Action Model. (The procedure for associating a connection resource with an HTML Action is discussed in detail in Chapter 4 on page 35.)

HTTP Authentication Types

Three main types of authentication are used in "secure" HTTP communication: Basic Authentication, Digest Authentication, and NTLM (NT LAN Manager). Composer offers three types of HTTP Connection Resources, corresponding to these three authentication protocols.

Basic Authentication

Basic Authentication is the most common type of HTTP authentication. If an HTTP client, such as a web browser, requests a page that is part of a protected realm, the server responds with a *401 Unauthorized* status code and includes a WWW-Authenticate header field in its response. This header field must contain at least one authentication challenge applicable to the requested page. The client then makes another request, this time including an Authentication header field which contains the client's credentials. If the server accepts the credentials, it returns the requested page. Otherwise, it returns another *401 Unauthorized* response to inform the client the authentication has failed.

One weakness of this type of scheme is that the user's credentials are transmitted "in the clear" and hence are susceptible to appropriation by eavesdroppers.

Digest Authentication

Digest Authentication works similarly to Basic Authentication, except that the user's credentials are encrypted before being sent over HTTP. Of course, merely encrypting a password before sending it over an open line doesn't add much safety, because the encrypted password can still be sniffed by a malicious program or individual and reused later. To keep this from happening,a host that supports Digest Authentication is required to generate a unique transaction ID value that can be associated with a given client-host session, and this unique per-transaction value—called a *nonce*—must be transmitted to the client and back from client to host as part of the authentication challenge. The significance of the nonce is that it can be used exactly once. If a hacker tries to reuse a stolen user-credential/nonce combo, authentication will fail since the nonce will be recognized by the host as having already been used. It is the host's responsibility to generate nonces and keep track of their use. Therefore you needn't concern yourself with this aspect of authentication when setting up an HTTP DigestAuthentication connection resource.

NTLM Authentication

NTLM (or NT LAN Manager) Authentication is a Microsoft-proprietary authentication protocol. It involves transmission of a hashed key to the server; rehashing of the key to a new value, which is nonce-appended and sent back to the client, and resending (by the client) of a newly rehashed key. The hash algorithms on each end are different, and since a nonce is involved (see above), the scheme is relatively secure against replay attacks.

NTLM authentication is often encountered at the local proxy level as well as at the target website or remote host. If you are going out through an NTLM-protected proxy, you will want to set your NTLM options in the **Tools > Configuration** dialog as shown earlier in the section called "Proxy Server Settings". If you will be visiting a web site that issues NTLM challenges, you will need to set up an HTTP NTLM Authentication connection resource as described below.

HTTP Basic Authentication Resource

The **HTTP Basic Authentication** connection type is provided as a means of letting you specify a security certificate and password info for access to a secure site. You will typically create one HTTP Basic Authentication connection resource for each secure site you visit. Later, you will assign the appropriate connection resource to each HTML Action that requires secure site access, per the description on page 35.

➢ **To create an HTML Connection Resource for a secure site:**

1   From the Composer **File** menu, select **New> xObject**, then open the **Resource** tab and select **Connection**.

NOTE: Alternatively, you can highlight **Connection** in the Composer window category pane, right click your mouse button, then select **New**.

The "Create a New Connection Resource" Wizard appears.

2   Type a **Name** for the connection object.

3   Optionally, type **Description** text.

4   Click **Next**.

5   Select **HTTP Basic Authentication Connection** from the Connection Type pull down menu.

6   Enter a **User ID** and **Password**. These are not actually submitted during the establishment of a connection. They are simply defined here (the password is encrypted). The user will have access to UserID and Password variables from ECMAScript, allowing them to map UserID and Password as values into the screen. This way, no one ever sees the passwords.

7   If the site requires a client-side certificate:

  ◆   Choose a **Client Certificate** by clicking on the **Browse** button and selecting the certificate file you want to use for this service connection.

  ◆   Choose a **Client Private key** by clicking on the **Browse** button and selecting the client key file for security.

  ◆   For steps above, please see *Appendix A* of this document for more detailed instructions about Digital Certificates.

  ◆   Enter the **Password** for the **Private key**. Private key is another level of security for the owner of the Client Private Key.

8   Enter a **Connection Timeout** value in seconds. This represents the maximum amount of time that your component will wait for the web page to download. If a connection is not established or the page doesn't download in the time allotted, an exception is thrown.

9 Select the **Default** check box if you wish for this connection resource to be the first one shown in the pulldown list in the HTML Action setup dialog (page 35) from this point on.

10 Click **Finish.** The connection resource is created.

HTTP Digest Authentication Connection Resource

The **HTTP Digest Authentication** connection type is provided as a means of letting you specify a username, password, and (optionally) a security certificate for access to a secure site that uses Digest Authnetication as described above. You will typically create one HTTP Digest Authentication connection resource for each secure site you visit. In building your component, you will assign the appropriate connection resource to each HTML Action that requires secure site access, per the description on page 36.

➢ **To create an HTTP Digest Authentication Connection Resource:**

See the procedure for the **HTTP Basic Authentication Connection Resource**, above. The dialogs are the same except for the words Basic and Digest.

HTTP NTLM Authentication Connection Resource

The **HTTP NTLM Authentication** connection type is provided as a means of letting you specify the credentials needed for access to a secure site that uses NTLM Authnetication.

You will typically create one HTTP NTLM Authentication connection resource for each secure site you visit. In building your component, you will assign the appropriate connection resource to each HTML Action that requires secure site access, per the description on page 36.

➢ **To create an HTTP NTLM Authentication Connection Resource:**

1 From the Composer **File** menu, select **New> xObject**, then open the **Resource** tab and select **Connection**.

NOTE: Alternatively, you can highlight **Connection** in the Composer window category pane, right click your mouse button, then select **New**.

The "Create a New Connection Resource" Wizard appears.

2 Type a **Name** for the connection object.

3 Optionally, type **Description** text.

4 Click **Next**.

5   Select **HTTP NTLM Authentication Connection** from the Connection Type pull down menu.

6   Enter a **User ID** and **Password**.

7   Enter the **Domain** name for the site (as in "http://www.domain.com").

8   If the site requires a client-side certificate:

• Choose a **Client Certificate** by clicking on the **Browse** button and selecting the certificate file you want to use for this service connection.

• Choose a **Client Private key** by clicking on the **Browse** button and selecting the client key file for security.

NOTE: For steps above, please see *Appendix A* of this document for more detailed instructions about Digital Certificates.

• Enter the **Password** for the **Private key**. Private key is another level of security for the owner of the Client Private Key.

9   Enter a **Connection Timeout** value in seconds. This represents the maximum amount of time that your component will wait for the web page to download. If a connection is not established or the page doesn't download in the time allotted, an exception is thrown.

10 Select the **Default** check box if you wish for this connection resource to be the first one shown in the pulldown list in the HTML Action setup dialog (page 36) from this point on.

11 Click **Finish.** The connection resource is created.

Creating XML Templates for Your Component

In addition to a connection resource, an HTML Component may also require that you have already created XML templates so that you have sample documents for designing your component. See Chapter 5, *Creating XML Templates* in the *exteNd Composer User's Guide* for more information.

3 Creating an HTML Component

Before Creating an HTML Component

As with all exteNd components, the first step in creating an HTML Component is to specify any XML templates needed. For more information, see *Creating a New XML Template* in the *Composer User's Guide.*

Once you've specified the XML templates, you can create a component, using the template's sample documents to represent the inputs and outputs processed by your component.

➢ **To create a new HTML Component:**

1 Select **File>New> xObject** then open the **Component** tab and select **HTML**. The Create a New HTML Component Wizard appears.


2 Enter a **Name** for the new HTML Component.
3 Optionally, type **Description** text.

4 Click **Next**. The XML Property Info panel of the New HTML Component Wizard appears.
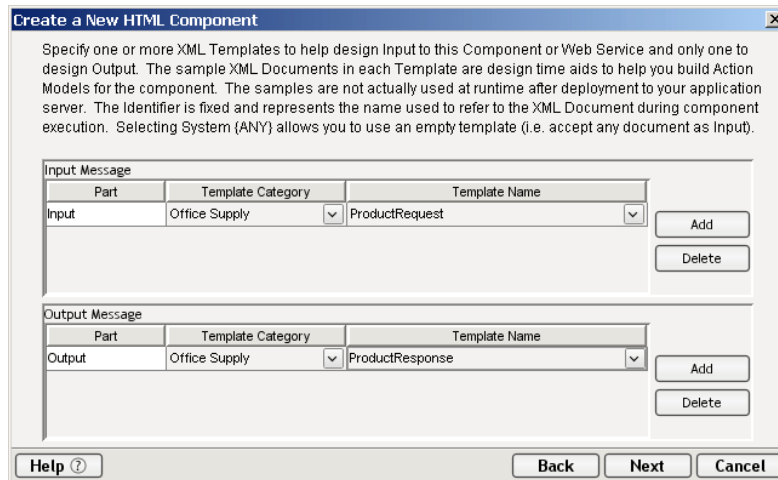
5 Specify the Input and Output templates (also called Messages).

◆ Type in a name for the template under **Part** if you wish the name to appear in the Component Editor as something other than "Input" or "Output."

◆ Select a **Template Category** if it is different than the default category. In the example above, the Office Supply Category has been selected.

◆ Select a **Template Name** from the list of XML templates in the selected **Template Category**. In the example above, Product Request has been selected for the Input Template.

◆ To add additional input XML templates, click **Add** and repeat steps 2 through 4.

◆ To remove an input XML template, select an entry and click **Delete**.

6 Select an XML template as an output using the same methods described in the previous step.

NOTE: You can specify an input or output XML template that contains no structure by selecting {System}{ANY} as the Output template. For more information, see "Creating an Output Message without Using a Template" in the *Composer User's Guide*.

7 Click **Next** to go the Temp and Fault XML template dialog.
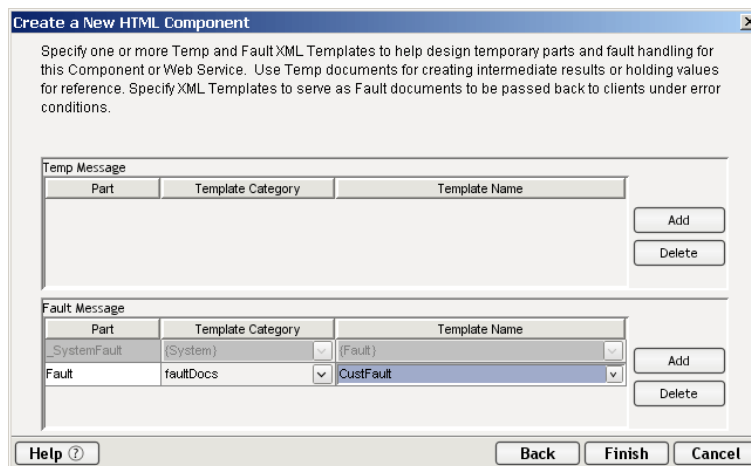
8 If desired, specify a template to be used as a scratchpad under the "Temp Message" pane of the dialog window. This can be useful if you need a place to hold values that will only be used temporarily during the execution of your component or are for reference only. Under the "Fault Message" pane, select an XML template to be used to pass back to clients when a fault condition occurs.

9 Under the "Fault Message" pane, select an XML template to be used to pass back to clients when an error condition occurs.

10 As above, to add additional temp or fault XML templates, click **Add** and choose a Template Category and Template Name for each. Repeat as many times as desired. To *remove* an XML template, select an entry and click **Delete**.

11 Click **Finish**. The component is created and the HTML Component Editor appears, as shown below.
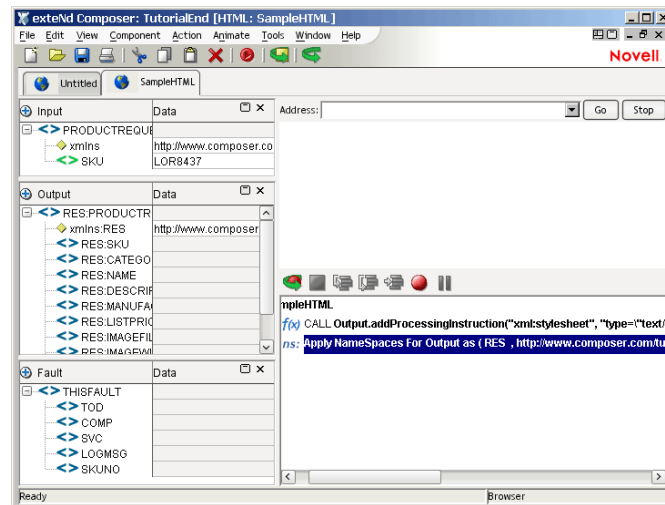
About the HTML Component Editor Window

The HTML Component Editor includes all the functionality of the XML Map Component Editor. For the HTML Connect, it contains mapping panes for Input and Output XML documents as well as an Action pane.

There are two key differences, however. The first is that the HTML Component Editor also includes a Native Environment Pane common to all Connects. It contains a web browser window with an address line at the top along with two buttons: Go and Stop. The second difference is the addition of an XML DOM called ScreenDoc to the component editor window. This DOM presents an XML document representation of each screen received from the website and is available for reference and creating mapping actions within the component. In addition, the ScreenDoc DOM is available in the expression builder, allowing the user to easily reference a screen field.
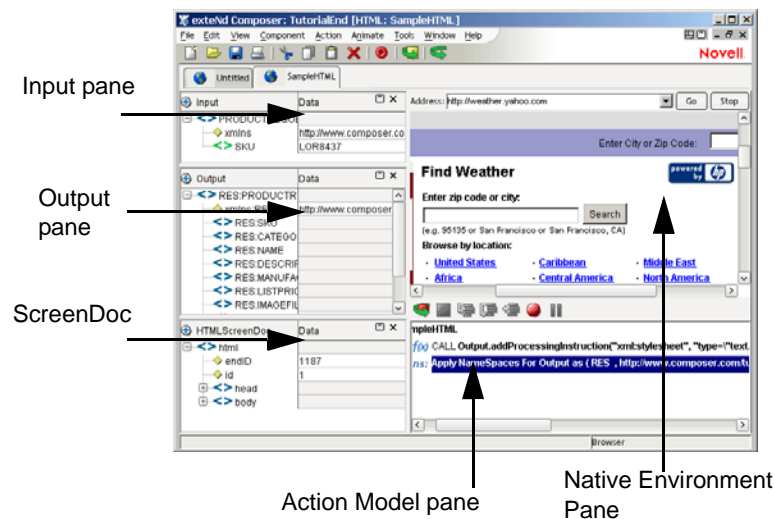
About the HTML Native Environment Pane

From the HTML Native Environment pane, you can perform the following:

- Map data from an Input XML document (or other available DOM) and use it as input for an HTML screen field or HTML form. For example, you could drag a SKU number from an input DOM into the part field of an HTML inquiry form, and return data associated with that part number, such as description and price.
- Map the data from the HTML screen and put it into an Output XML document (or other available DOM, e.g., Temp, MyDom, etc.).
- Build B2B services that interact with secure Web sites using SSL3 and Digital Certificates.

Except for "rich content" pages containing Flash, Quicktime movies, etc. (which will render in non-rich form; Composer will ignore the media elements), the HTML Native Environment Pane functions exactly as you would expect a web browser to function and allows you to do the same sorts of things.

About the ScreenDoc DOM

The ScreenDoc DOM is an XML document representation (actually XHTML after the Web page has been processed by Tidy) of the current web page displayed in the browser of the Native Environment pane. All Mapping actions to and from the screen display (including drag and drop) actually reference elements in the ScreenDoc DOM. This provides you with the advantage of being able to see and reference your familiar application screens while at the same time working with them as XML documents.

How it works

Each time a web page is received by the component, several things happen simultaneously:

- The original HTML Web page is displayed in the Native Environment pane.
- The original HTML is processed by Tidy, creating an XHTML version of the web page that is then displayed in the ScreenDoc DOM. You can now use any Action on the ScreenDoc DOM as you would any other XML document.
- An HTML Action is created (during Record mode only) containing the URI of the received Web page including any parameters sent to the host.

The end result of displaying the Web page as XHTML is that the ScreenDoc DOM can be quite large. Its use is primarily intended for finding hidden fields, verifying fields, parameter field values, and in cases where it is convenient, mapping from the ScreenDoc DOM to the Output DOM using Composer's drag and drop features.

NOTE: Normally it is much quicker and more efficient to map directly to and from the Native Environment pane using drag-and-drop instead of mapping to the ScreenDoc DOM. The XHTML in the ScreenDoc DOM displays all details of the Web page including attributes related to fonts, table attributes, etc. To avoid mapping this irrelevant data, use Drag and Drop when possible, dragging directly to and from the Native Environment Pane.

➢ Making the ScreenDoc DOM visible:

1   Select **View/Window Layout** from the Component Window Menu.

2   The Window Layout dialog appears and allows you to adjust the placement of the panels in the Window. Use the drop-down arrow in the four different fields, and select the placement of the Panes.
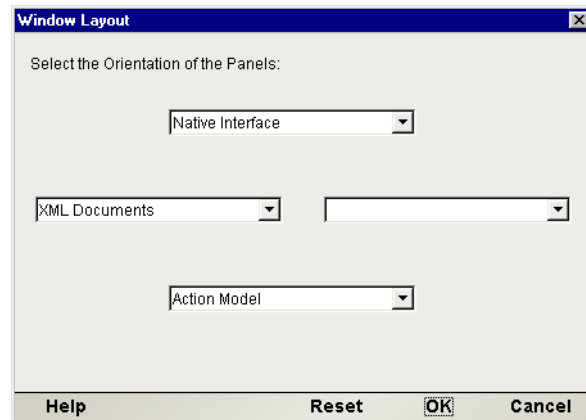
3   Click **OK** to close the dialog. Click **Reset** if you decide to change your settings.

➢ To arrange the view of the XML documents in the component editor:

1   Select **View/XML Documents>Show/Hide**

2   By using the directional buttons, you can move the Panes from the Invisible column to the Visible Column or vice versa. You can also choose the order in which visible selections appear on the screen.

3   Click **OK** to save your settings. Click **Reset** if you decide to change your settings.
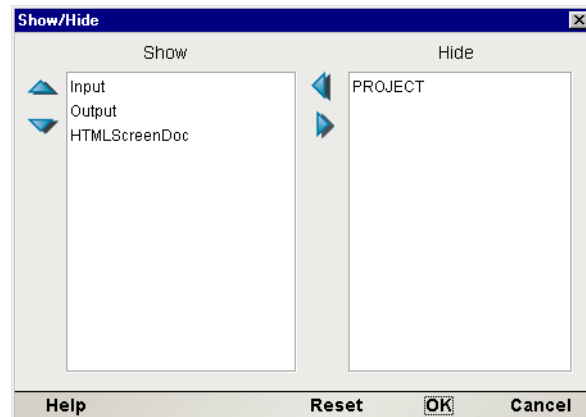
About HTML-Specific Menu Bar Items

When you are using the HTML Connect editing environment, Composer's main menus have certain commands that are specific to the HTML Connect.
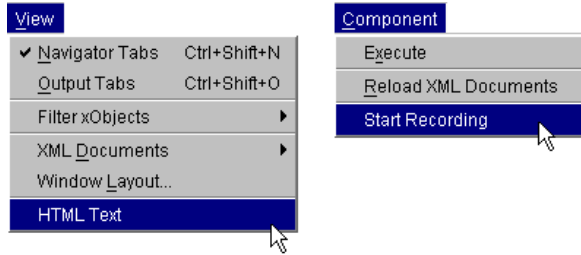
View Menu

HTML Text—This command brings up a window that displays the HTML code (source code) for the page that is currently displayed in the Native Environment Pane. This is similar to View/Source in Netscape or Internet Explorer. See below.
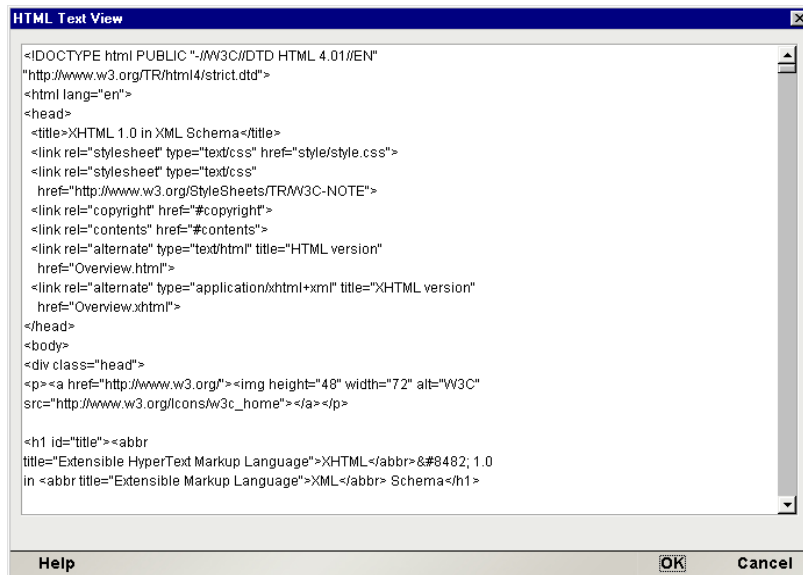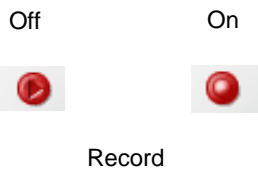

Component Menu

Start Recording—This command will result in Composer creating actions dynamically as you interact in real time with the Native Environment Pane.

To turn recording off, click the **Record** button (see next section). The button acts as a toggle.

About HTML-Specific Buttons

Record

The HTML Connect puts an additional tool button on the component editor tool bar: the **Record** button. The **Record** button results in Composer putting new actions in the Action Model as you interact with HTML screens.

Off   On

Record

Execute/Execute All

Next to the Record button, you will find **Execute/Execute All** buttons. These buttons allow you to run selected actions in your component, or the entire action model, respectively.

Execute   Execute All

Execute/Execute All

4 Performing HTML Actions

About Actions

An *action* is similar to a programming statement in that it takes input in the form of parameters and performs specific tasks. Please see the chapters in the *Composer User's Guide* devoted to Actions.

Within the HTML Component Editor, a set of instructions for processing XML documents or communicating with non-XML data sources is created as part of an *Action Model*. The Action Model performs all data mapping, data transformation, data transfer between the Web site screen and XML documents, and data transfer within components and services.

The HTML Connect has two HTML-related actions: HTML Action and Set Frame. The HTML action accesses a URI and encodes parameters for the URI Actions. HTML Actions include HTTP Get and HTTP Post. These action types are referred to as methods that request data. HTTP Get is a method that gets the file with a query consisting of the specified form-data. The data is urlencoded, turned into a string, appended to the end of the URL, and then sent as a query string.

There can be multiple frames on a Web page and each frame has its own document. In order to set the context of the frame, the Set Frame Action, which is automatically generated, allows you to set the context of the selected frame, so that the DOM reflects the contents of that frame.

HTTP Post is a request type that posts data to the specified URI using specific headers. HTTP Post sends larger amounts of information to the server to process. Of course, this is dependent upon the embedded form on the Web page. The data is first URIencoded and then turned into a string of the form used. The response to an action can be a redirect. A redirect is automatically handled and requires no interaction from the user.

When Composer receives an HTTP content stream consisting of HTML, it will render it at design time in the Native Environment Pane. Using a checkbox in the HTML Action dialog, you can turn JavaScript support on or off. If you turn it on, you will be able to interact with scripted pages, including forms that use JavaScript event handlers. (This is subject to certain limitations. See "JavaScript versus ECMAScript" and "User Agent Info" later in this chapter.)

Recording an HTML Session

The HTML Component differs from other components because a major portion of the Action Model is built for you automatically. This happens as you interact with a remote site in a live HTTP session in Composer, while you are using Composer's "recording" capability to capture your activities as a set of actions in the Action Model. In other components, you must manually create actions in the Action Model, which then perform mapping, transformation, and other tasks. When you create an HTML Component, you essentially *record* the requests and responses to and from the website, which are captured as a combination of HTML Actions, Map Actions, and Function Actions in the Action Model pane.

➢ **To record a simple HTML session:**

1   Create an HTML Component per the instructions in "*How to Create an HTML Component*" in Chapter 3 of this Guide. In creating the HTML Component shown in this example, the Office Supply Product Request and Product Response templates were selected for Input and Output respectively. Once created, the new HTML Component appears in the HTML Component Editor window.

2   Enter an address in the URI field. In the example, the address used is: **http://localhost/XCTutorial/loriconproductsearch**. Click on the **GO** button or press the **Enter** key on your keyboard.


Record button   Target URI
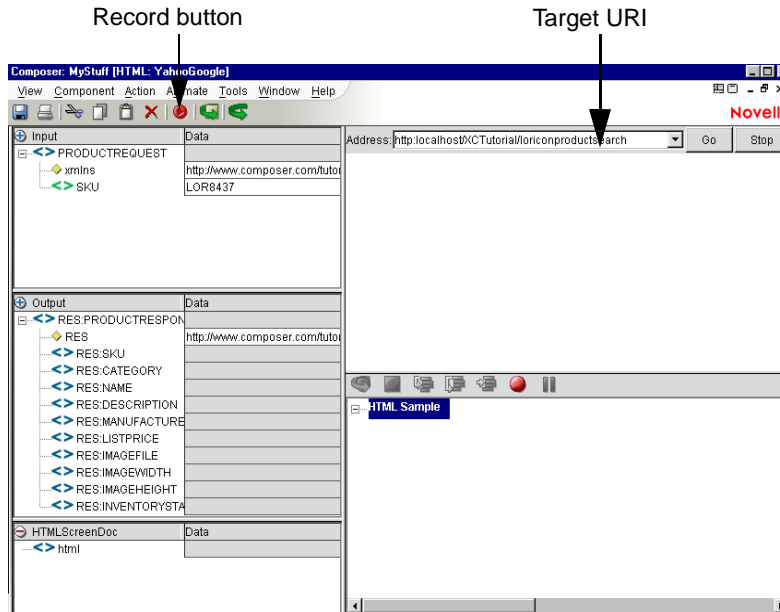
3   Click the **Record** button and the current Web page is captured as an HTML Action in the Action Model (see below).
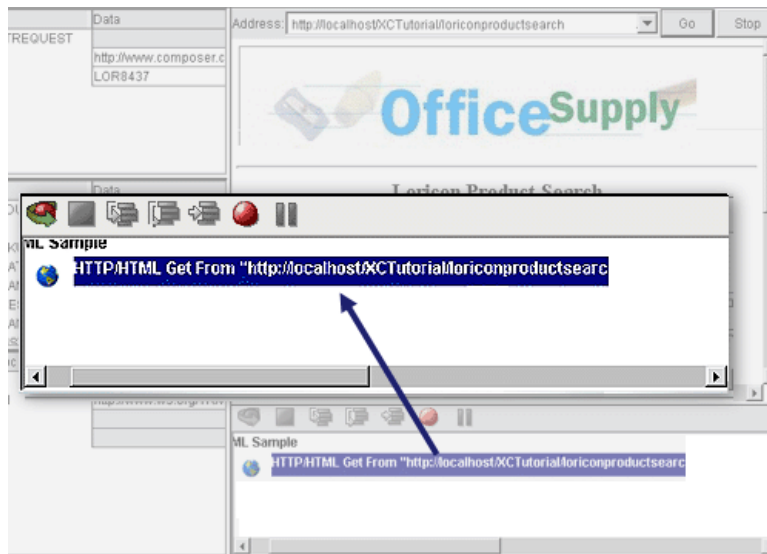
HTML Action is automatically added to Action Model.

4   Drag the SKU information "LOR8437" from the Input DOM into the entry field on the form displayed in the Native Environment Pane and drop it into the field. A Map Action will be created in the Action Pane.
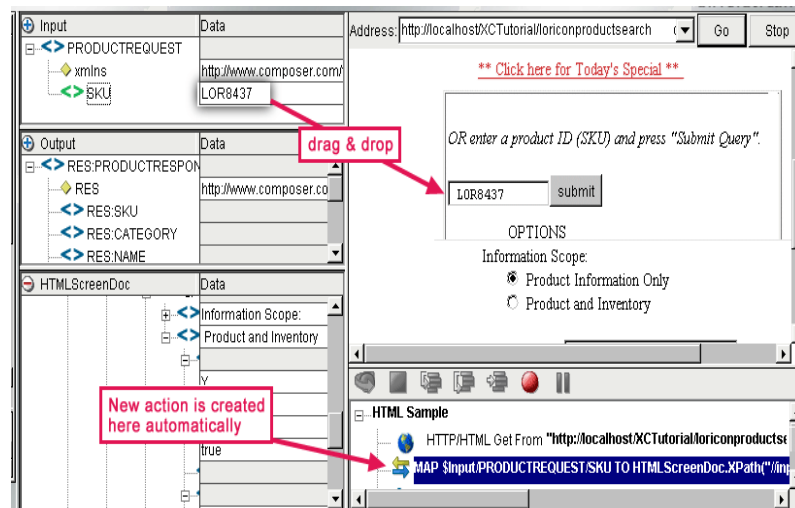
5   In the Native Environment Pane, after entering the SKU, deselect the default option and click on the radio button for **Product and Inventory**.
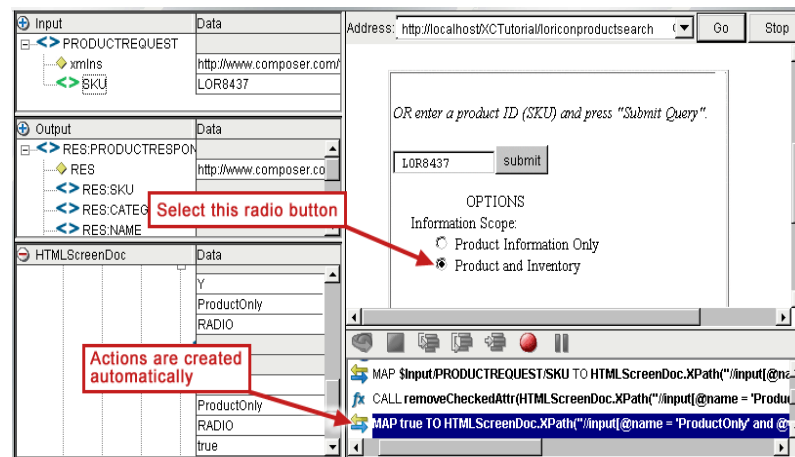

NOTE: A *Map Action* is automatically created in the Action Pane when a new selection is made when interacting with the form in the Native Environment pane. A *Function Action* is created in the Action Pane when you deselect a previous selection within the form. For example, in this sample, you deselected the default radio button and selected the Product and Inventory and clicked on that radio button. In this case, both types of Actions, Function and Map, are created in the Action Pane. Please refer to Appendix D for a complete listing of actions created as a result of interacting with the HTML visual controls.

6   Click on the checkbox to **Omit Product Description**.

NOTE: New actions will be created in the Action Model as you interact with form controls in the Native Environment Pane. In this example, you clicked in the checkbox and selected Omit Product Description.

7   Click on the **Submit** button. The new Web page appears with the search information that was requested.

8   Drag and drop an element from the search results screen to the Ouput DOM: For example, drag "$410" into the LISTPRICE field in the Output DOM. The data you drag and drop appears in red.

9   If you wish, you can continue to drag and drop data elements from the search results screen to the desired field in the Output DOM until complete. Each time an element is dragged from the search results page to the Output DOM an action is recorded in the Action Model pane.

NOTE: Notice that the HTML ScreenDoc DOM, which mirrors the Web page in the Native environment, is being updated as each action occurs. You can see more details by expanding the HTMLScreenDoc tree.

10 Click the **Record** button to turn off the recording mode.

11 Start animation by clicking on the Animate icon and stepping through the actions in the Action Pane by repeatedly clicking the Step Into icon. (See below.)
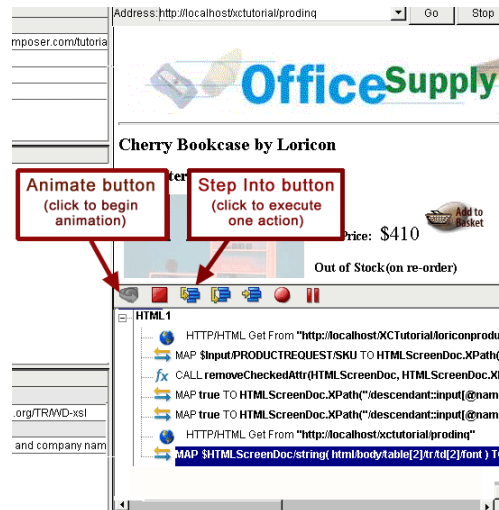
12 When the process is complete, a small dialog appears. Click **OK**.
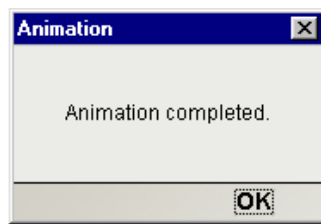
On the following screen, notice that the Output DOM contains only the "$410" that was requested from the Web page.
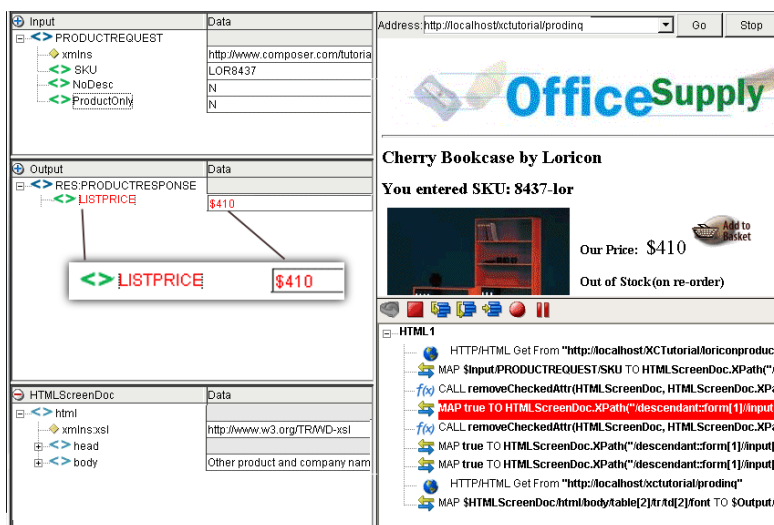
Recording an HTML Session using Frames

If the Web page you are using contains multiple frames, HTML Connect will create and record a Set Frame Action in the Map Screen when you click from frame to frame or drag and drop data into a DOM. The HTML Screen Doc will show a DOM for this frame.

➢ **To record a HTML Session using Pages Containing Frames**

1   Create an HTML Component per the instructions in How to Create an HTML Component in Chapter 3 of this Guide. In creating the HTML Component shown in this example, simple templates are used for Input and Output respectively. Once created, the new HTML Component appears in the HTML Component Editor window.

2   Enter an address in the URI field. In this example, the address used is:
    http://localhost/XCTutorial/Silverstream/Objectstore/General/LoriconHome.html

3   Click on the **GO** button or press the **Enter** key on your keyboard.

4   Click the **Record** button and the current Web page is captured as an HTML Action in the Action Model.
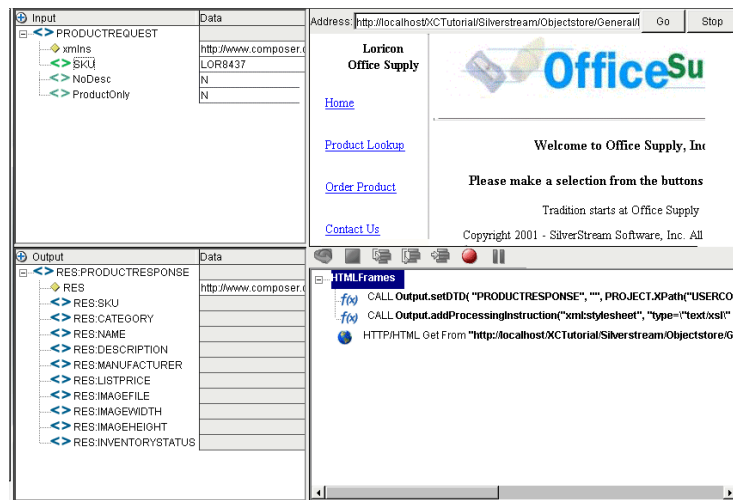
In this example, click on Product Lookup. In the Map Action Pane, you can see that a Set Frame action was created to show that you now entered a different frame. The actions following are then recorded.
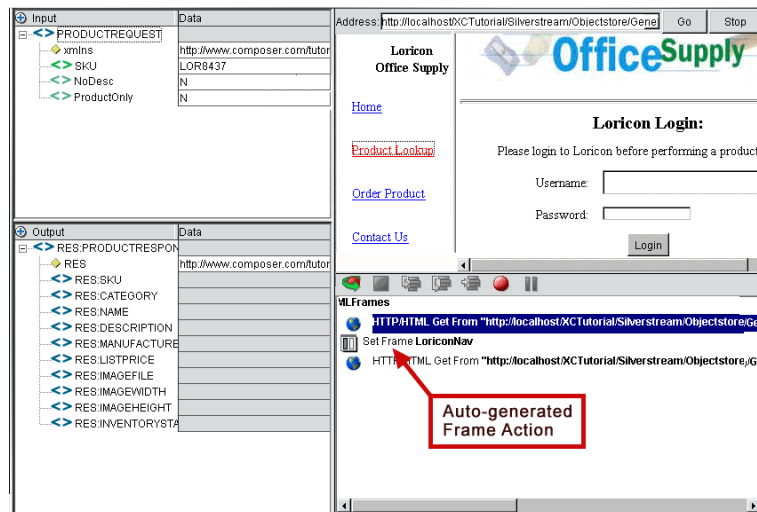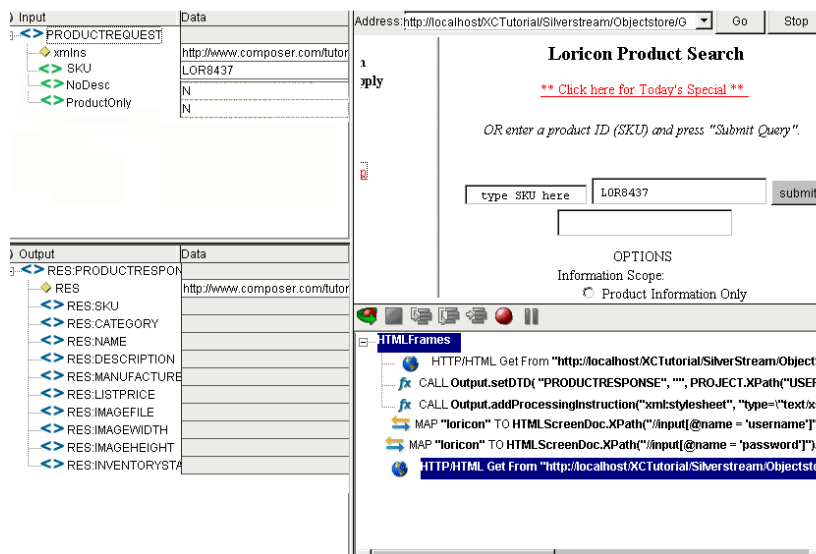
5   After making your selection, a Login Screen appears requesting Username and Password.
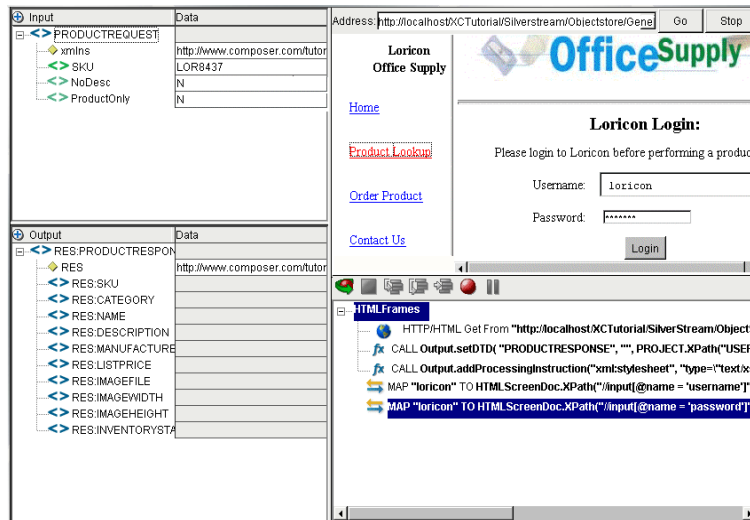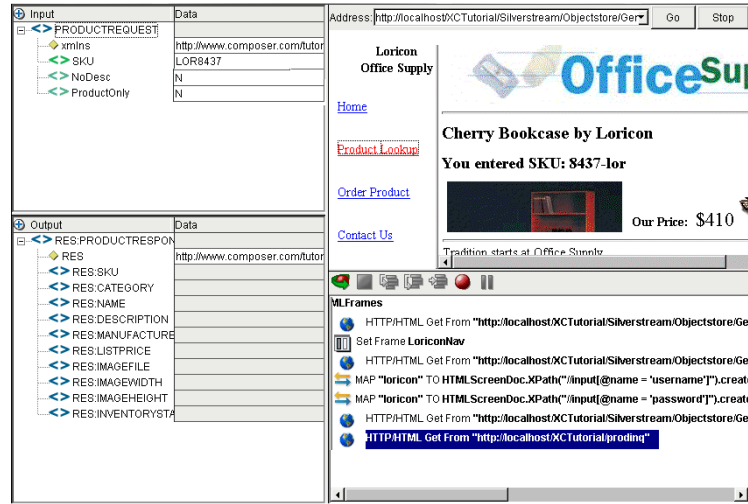

6   Enter your Username and Password. In this example, both the Username and Password is **loricon**. Click on the **Login** button. Notice the actions created in the Map Action pane.

7   **Enter the SKU**, which for our example is **LOR8437** and click on the **Submit** button.

8 Once you click the Submit button, your response to your request will appear in the Navigation Pane (see below).
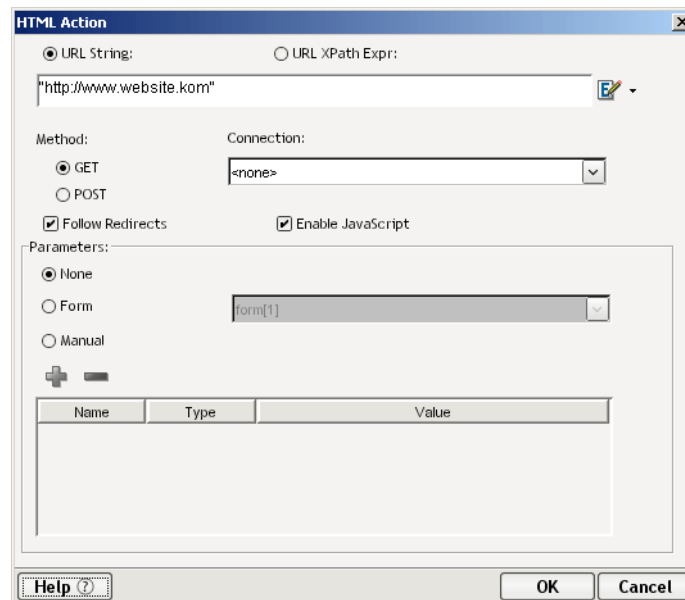

Editing an HTML Action

Once an HTML Action has been created (in Record mode), you can edit various parameters associated with it. See below.

➢ **To edit links and parameters in the HTML Action**

1 Doubleclick an existing HTML Action in the Action Model. The HTML Action dialog appears, as shown below.

2 Enter an address of a Web site screen you want to navigate to in the **URL** field. You can either type the string or click on the Expression icon and create an ECMAScript expression that specifies the URL of interest. The URL can also be obtained via XPath. (Use the radio button provided.)

NOTE: If you enter the URL directly (as shown above), in ECMAScript mode, be sure to surround it with double quotes.

3　Select the **Method** type by click on the radio button for GET or POST. You will most often choose GET.

4　Optionally select an HTTP Connection Resource from the pulldown list. This list is prepopulated with the names of any HTTP Connection Resources that exist in the current project.

NOTE:　You would normally specify an HTTP Resource only when using a secure connection or a connection on which you wish to set a timeout value.

5　Click the **Follow Redirects** checkbox if you want your HTML Action to honor redirections at runtime by default.

6　Optionally check the **Enable JavaScript** checkbox if the site you are going to requires a JavaScript-enabled browser.

NOTE:　Composer does not implement every JavaScript extension supported by every browser type. Hence, you may run into situation where a particular site uses scripts that are problematic in Composer. This can only be uncovered by testing.

7　The **Parameters** section contains three radio buttons: None, Form and Manual. Select one of the radio buttons.

◆　**None** selects no parameters.

◆　**Form** allows you to select a form by name from the pulldown list. The list will show every form embedded in the HTML page. Selecting a form will cause the fields contained in that form to show up (by Name, Type, and Value) in the table at the bottom of the dialog.

◆　**Manual** allows you to add a Name, Type, and Value for a new form field for any form. It also allows you to remove any field from the selected form. (These changes cause the ScreenDoc DOM to update appropriately.) After selecting the **ADD** button, enter information for Name, Type, and Value for the new field. The **DELETE** button allows you to delete a field from the list if you wish to do so. When you click in the Value area, the Expression icon appears. Click on the icon and the Expression Builder dialog appears, which allows you to easily create your expression using picklist items.
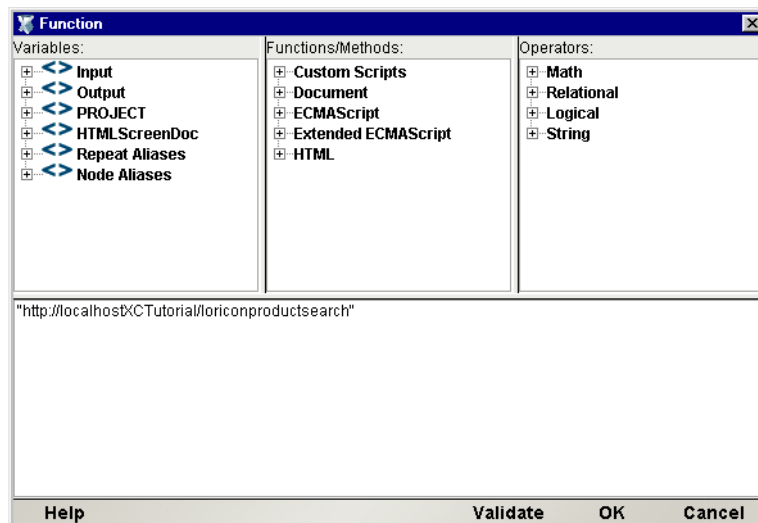
NOTE:　Because this is an ECMAScript string, it must be enclosed in double-quotes.

8　Click **OK** when finished and return to the HTML Action dialog.

9　Close the HTML Action dialog by clicking **OK**.

10　**Save** the component.

Editing a Previously Recorded Action Model

You will undoubtedly encounter times when you need to edit a previously recorded action model. Unlike editing other components, editing an HTML Component requires extra attention. When an HTML Component executes, it plays back a sequence of actions that expect certain Web pages and data to appear in order to work properly. So when editing a component you must be careful not to make the action model sequence inconsistent with the Web page execution sequence you recorded earlier.

In general, to ensure successful edits, the following recommendations apply:

◆ Do not cut or copy HTML Actions and paste them into other locations in your action model.

◆ Carefully check and edit individual Map actions that interact with the HTMLScreenDoc DOM after copying and pasting any Map actions.

◆ Use Composer's drag-and-drop features to add new Map actions that interact with the screen.

The safest procedure when editing or adding actions is to Animate to the line of interest in your Action Model, pause animation, and turn on Record mode. This will prevent your Action Model from getting out of sync with the proper ScreenDoc DOM and /or fields within a specific ScreenDoc DOM.

Changing an Existing Action

The following procedure will explain how to change an existing action in a previously recorded session.

➢ **To Change an existing action in a previously recorded Action Model:**

1 Open the component that includes the previously recorded Action Model that you'd like to edit. The component appears in the HTML Component Editor window.
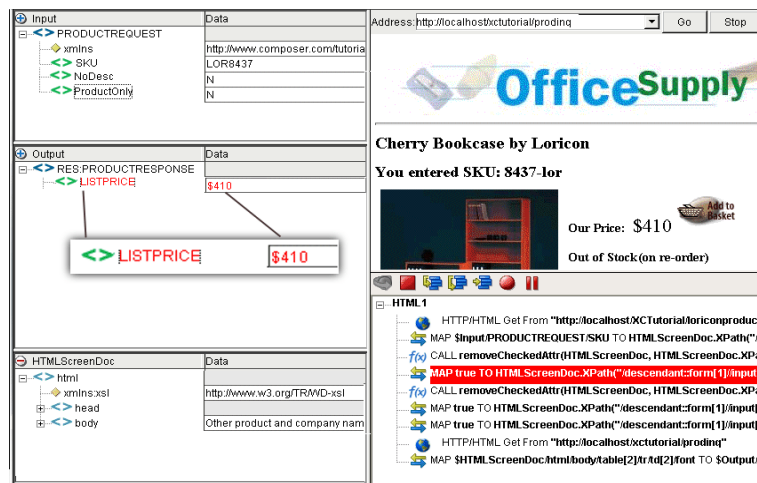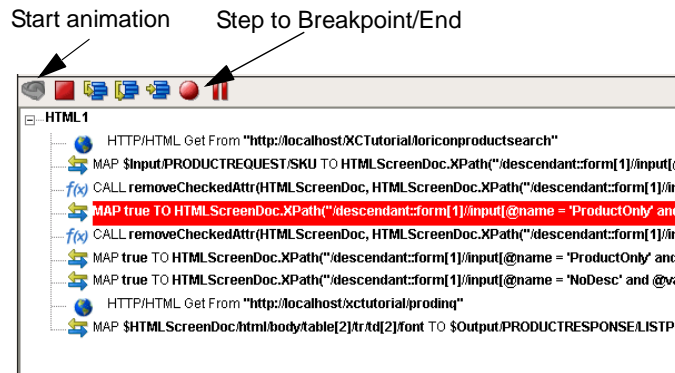

2 Locate the action in the Action Model where you'd like to make your edit and highlight the action.
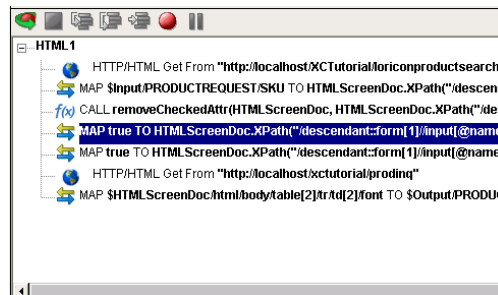
3   Click the **Toggle Breakpoint** button (or press F2). The highlighted action becomes red.

Toggle breakpoint

4   Click the **Start Animation** button. The animation tools become active.
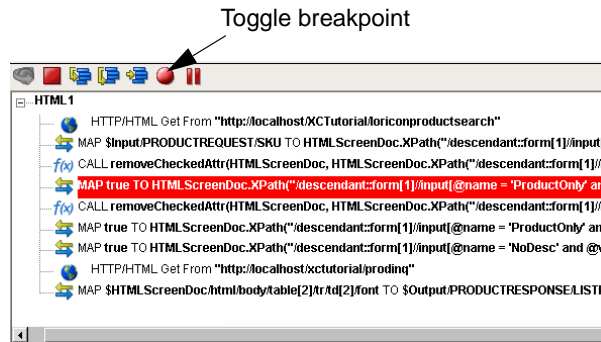
Start animation   Step to Breakpoint/End

5   Click the **Run to Breakpoint/End** button. The Action Model executes all of the actions from the beginning to the breakpoint you set in step 3 above and appears as shown.

6   Click the **Pause** button on the animation tool bar.

7   In the Component Editor tool bar, click the **Record** button.

8   Execute any additional actions that you'd like to make to the Action Model.

9   Click the **Record** button a second time to turn *off* the recording mode.

10 Select **File**, then **Save**, or click the **Save** button on the Component Editor tool bar.

11 Follow the instructions in "Using Animation Tools" to test your component.

Adding a New Action

The following procedure explains how to add a new action in a previously recorded session.

➢ **To Add a Action to a previously recorded Action Model:**

1   Open the component that includes the previously recorded Action Model that you'd like to add an action in. The component appears in the HTML Component Editor window.
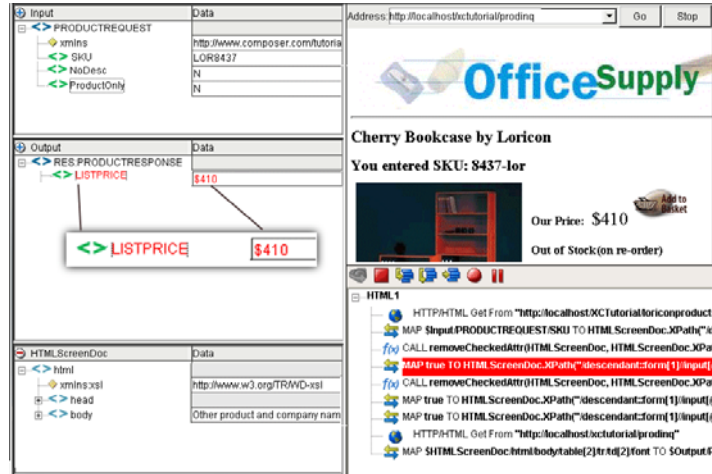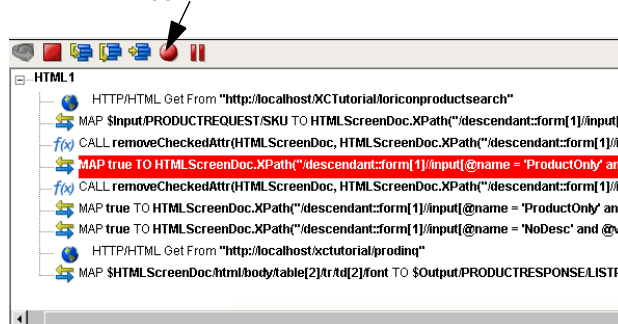

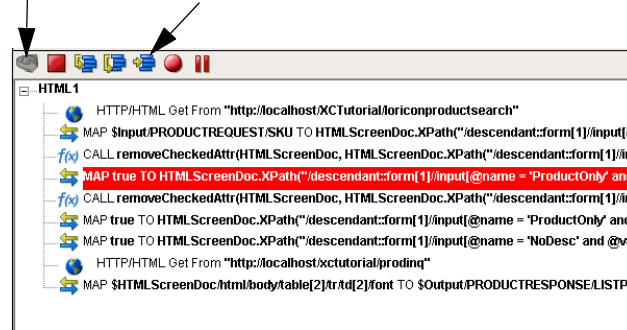2   Locate the action in the Action Model where you'd like to make your addition and highlight the action.
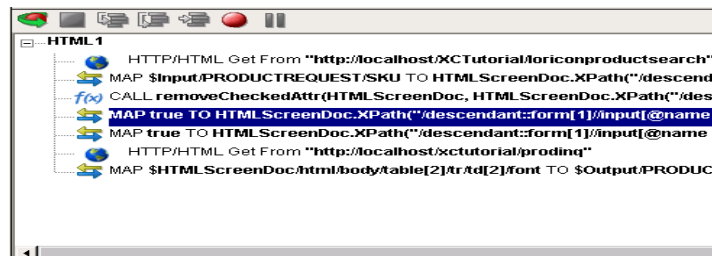
3   Click the **Toggle Breakpoint** button (or press F2). The highlighted action becomes red.
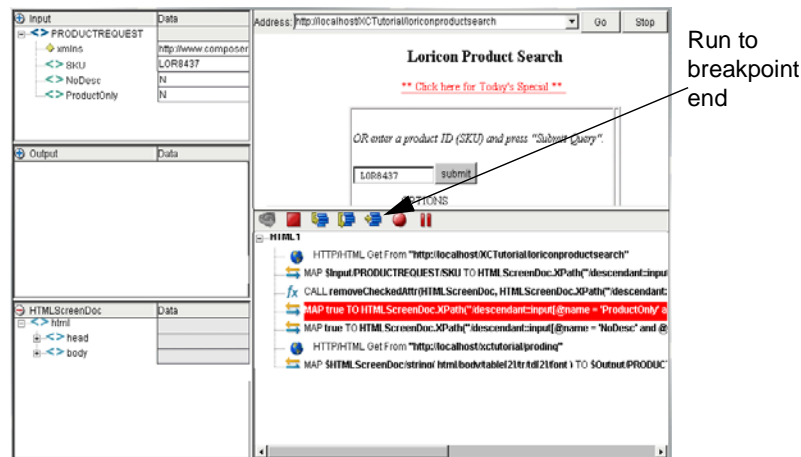
Toggle breakpoint

4   Click the **Start Animation** button. The animation tools become active.

Start animation   Run to Breakpoint/End

5   Click the **Run to Breakpoint/End** button. The Action Model executes all of the actions from the beginning to the breakpoint you set in step 3 above and appears as shown.

Run to
breakpoint
end

6   Click the **Pause** button on the animation tool bar.

7   In the Component Editor tool bar, click the **Record** button.

8   Highlight the action line and RMB on **Action > New Action > Html Action**. (Or just doubleclick the line.) The HTML Action dialog appears.
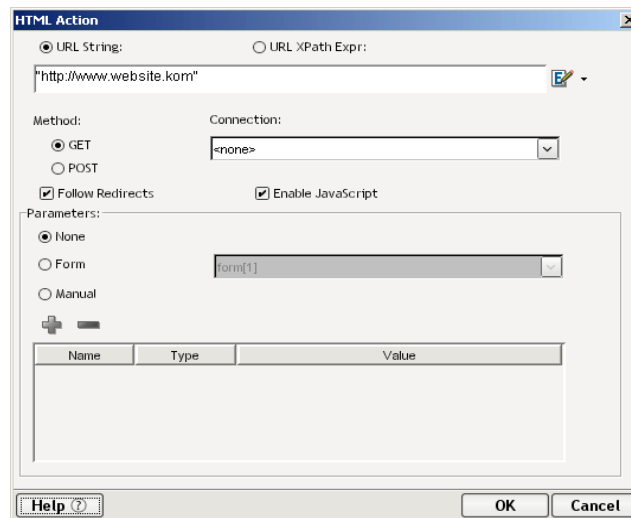
9   After entering completing the required fields on the screen, click the **OK** button. The new action will be added directly under the one highlighted.

10 Click the **Record** button a second time to turn *off* the recording mode.

11 Select **File**, then **Save**, or click the **Save** button on the Component Editor tool bar.

12 Follow the instructions in "Using Animation Tools" to test your component.

13 The action will be added directly after the highlighted line.

Deleting an Action

The following procedure explains how to delete an action in a previously recorded session.

➤ **To Delete an Action to a previously recorded Action Model:**

1 Highlight the action line that you want to delete and click on the RMB and select Delete from the menu. You may also highlight the line and press the Delete button on your keyboard.
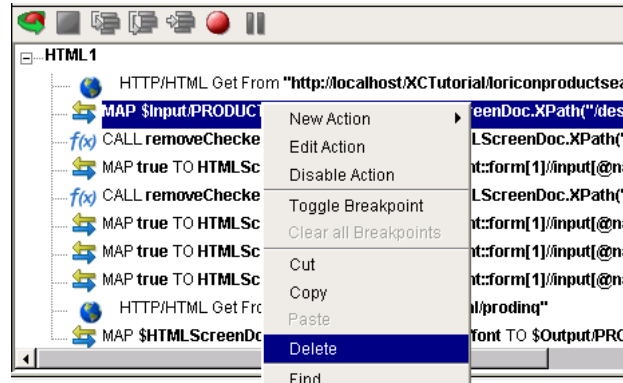

Executing Your HTML Component

Composer includes animation tools that allow you to test your component. On the HTML Component Editor tool bar you'll find the **Execute** button, which allows you to execute the entire Action Model and verify that your component works as you intend.

➤ **To execute an HTML Component:**

1 Open an HTML Component. The HTML Component Editor window appears.
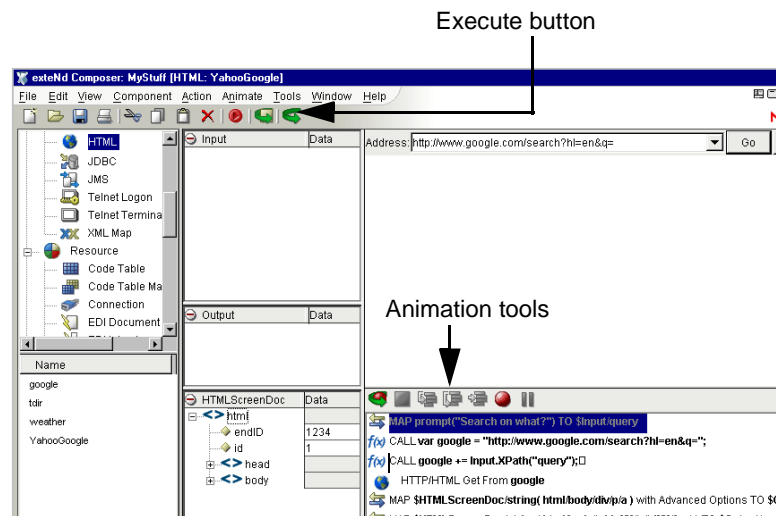
2   Select the **Execute** button. The actions in the Action Model execute and, when complete, a message appears.
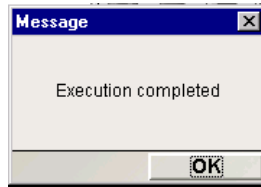
3   Click **OK**.

4   From the **View** menu, select **Expand XML Documents**. This expands all of the parents, children, data elements, etc. of the XML Documents, which allows you to see the results of the executed component. If you do not expand the XML Documents, you won't see if the data you wanted to move from the HTML environment made it to the Output DOM.

Using the Animation Tools

In the Action Model, you'll find animation tools that allow you to test a particular section of the Action Model by setting one or more breakpoints. This way, you can run through the actions that work properly, stop at the actions that are giving you trouble, and then troubleshoot the problem actions one at a time.

NOTE: The following procedure is a brief example of the functionality of the animation tools. For a complete description of all the animation tools and their functionality, please refer to the *exteNd Composer User's Guide*.

➢ **To run the animation:**

1   Open an HTML Component. The component appears in the HTML Component Editor window.
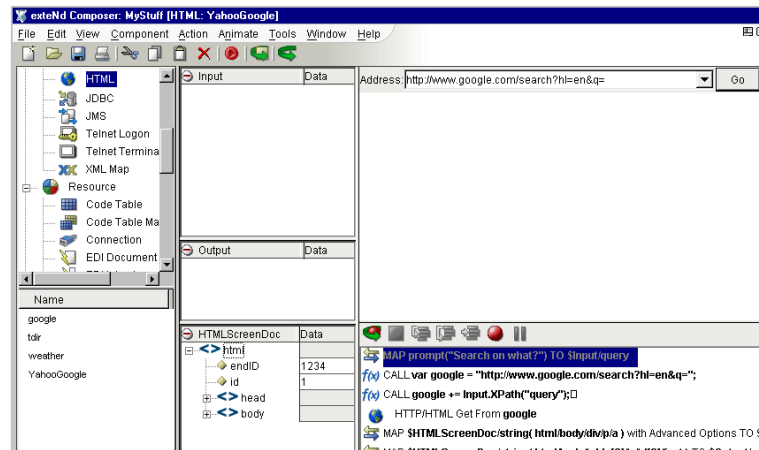
NOTE: Animation and Recording are mutually exclusive modes in the component. In order to record during animation, you must either pause or stop animation and then turn on Record mode.

2   Click the **Start Animation** button in the Action Model tool bar, or press F5 on the keyboard. All of the tools on the tool bar become active.

Step over

Step into   Run to breakpoint
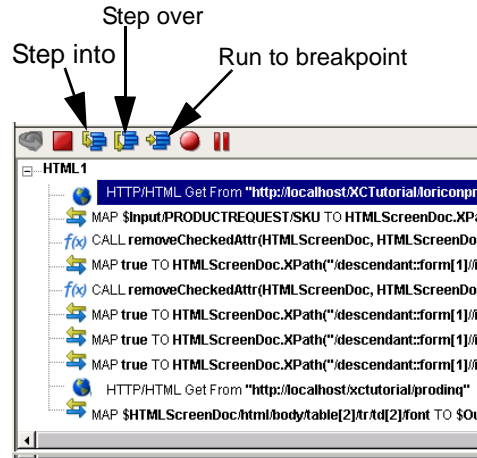
3   Click the **Step Into** button. The HTML Action will become highlighted. Click the **Step Into** button again. The first Map Screen action becomes highlighted.

Highlighted
action
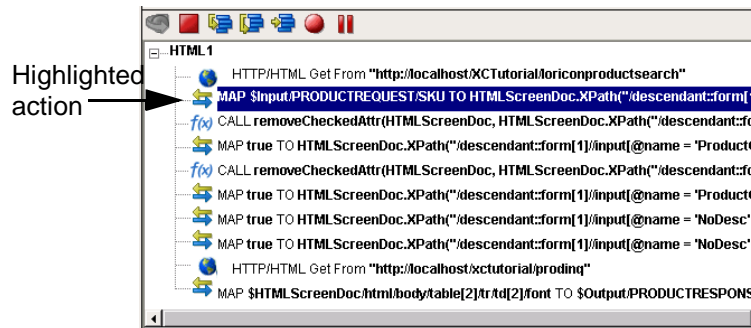
4   Click the **Step Into** button again. Notice in the Native Environment Pane that the SKU is filled in.
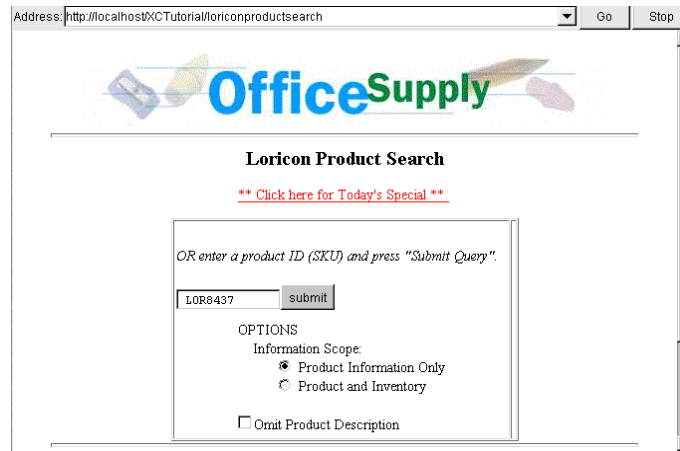
5   Click on the **Step Into** button again. Then click on the Step again and into you reach the Map Action.
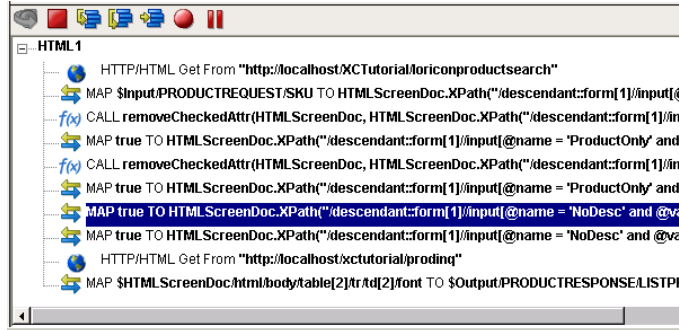
6   Notice that in the Native Environment pane the radio buttons changed.
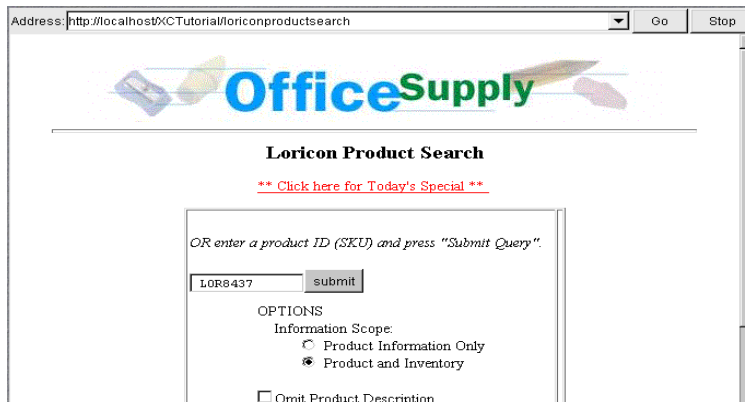
Click the **Step Into** button again.
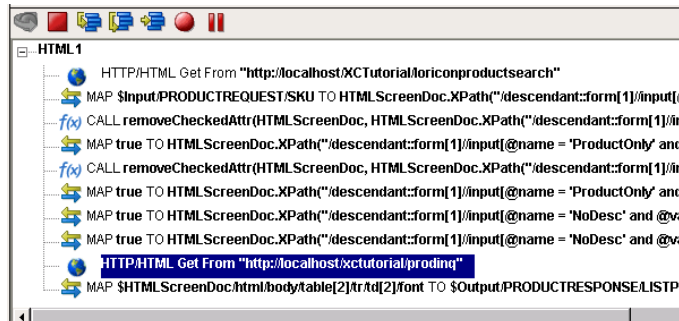
7   Notice the Native Environment Pane now displays the checkmark selection in the Omit Product Description.
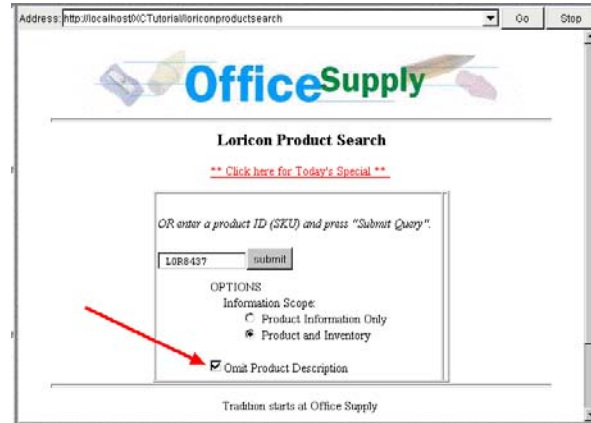
8   Click the **Step Into** button again. In the Action Model, the highlighted action is an HTML action.
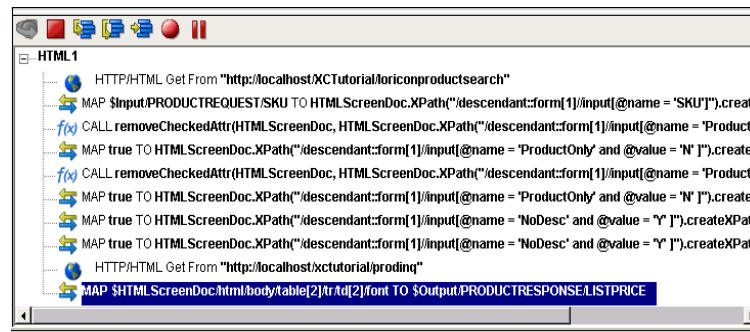
9   Click the **Step Into** button again. After stepping into the next action the Native Environment Pane will reflect the next Web page screen.

10 Click the **Step Into** button again.

11 Once complete, the following message appears.
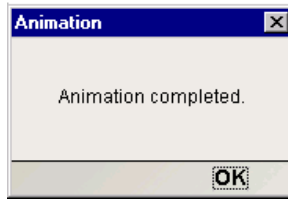
12 Click **OK**. Notice the Output DOM; it now contains the drag and drop information, "$410," in the ListPrice field. See below.
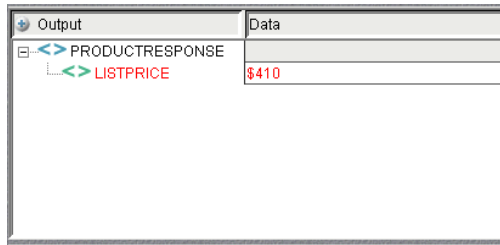
Using Other Actions in the HTML Component Editor

In addition to the Map Action, you have all the standard Basic and Advanced Composer actions at your disposal as well. The complete listing of Basic Composer Actions can be found in Chapter 7 of the *Composer User's Guide*. Chapter 8 contains a listing of the more Advanced Actions available to you. The XML Interchange Advanced Data Exchange action is particularly relevant to the HTML connect, so it is discussed in detail below.

Using the XML Interchange Action

The XML Interchange Action allows you to read or write an external XML document into a DOM into the component from a specific URI.

➢ **To use the XML Interchange Action:**

1 Create a new XML Map component and choose from the main Menu bar, **Action**>**New Action**> **Data Exchange**>**XML Interchange**.
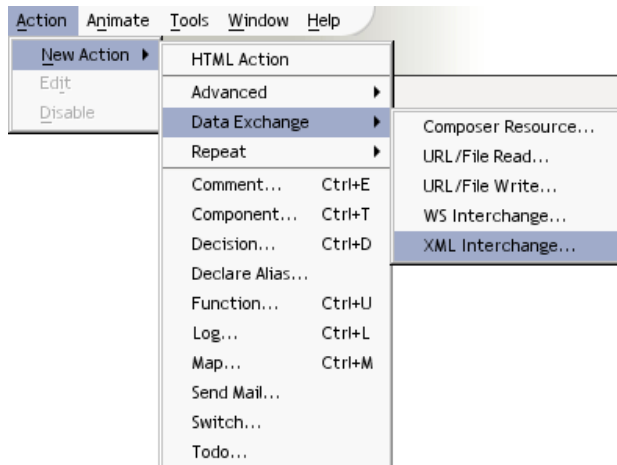
2   The dialog box appears.

.
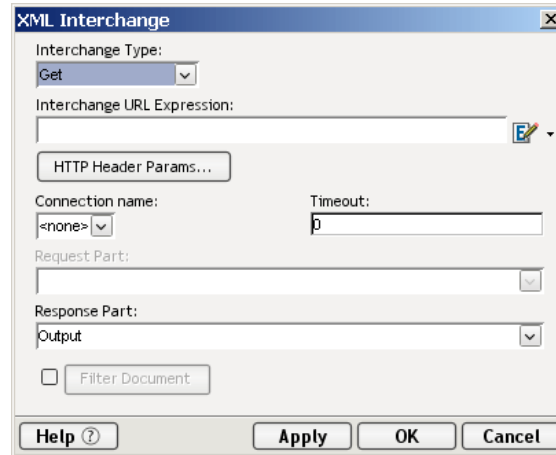
3   From the dropdown list, select the **Interchange Type**. You can select Get, Put, Post or Post with response.

4   Enter the Interchange URL Expression or click on the source expression icon.

5   If you want to add or delete a parameter and value to the Header, click on the **HTTP Header Params** button and the following dialog appears.
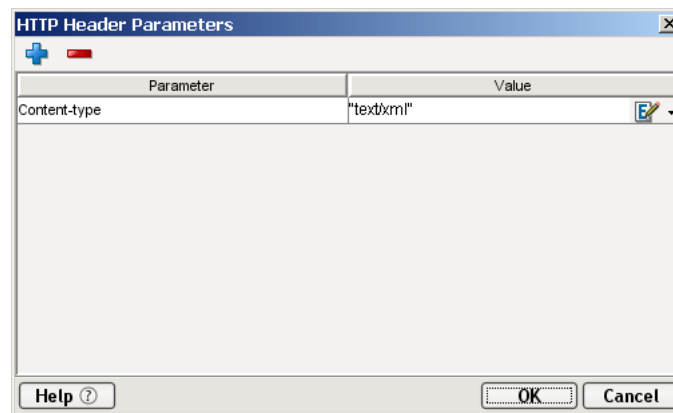
6   Click on the plus sign icon (+) to add a row or click on the minus (-) to delete a row. Click on the Source Expression icon to enter a header value expression.

7   Click **OK** and return to the HTTP Header Parameters dialog, click OK again and return to the XML Interchange dialog.

8   Select a **Connection Name** from the dropdown list.

9   Specify a Connection **Timeout** value (in seconds), or leave as zero. Whatever value you place here will override any value specified in your connection resource.

10   **Document Handle** field will default to Output. The name of this Document Handle will change according to the selection you made in the Interchange Type field.

11   Response Part Document Handle is active only when you select Post with Response from the Interchange Type dropdown list.

12   Click **OK** and a map action is then created dependent upon your selections.

Performance

XPath evaluations can take a significant amount of time when a DOM representation of an HTML page is large and complex. The amount of time spent in XPath evaluation can vary a great deal depending not only on page complexity, but the XPath processor used. The Xalan processor, which is the default XPath engine used by the exteNd Composer HTML Connect, employs a depth-first node-walking algorithm, which gives good performance on small to medium-sized HTML pages of low complexity. The Jaxen Xpath processor, on the other hand, uses a breadth-first algorithm, which gives better performance on large and/or complex DOMs.

The exteNd Composer HTML Connect allows you to choose the XPath engine you want to use. To change the processor, call the `setXPathProcessor()` method on the DOM object in question (usually `HTMLScreenDoc`), supplying a string argument of "xalan" (default) or "jaxen." For example, create a new Function action and enter an expression as follows in the Function dialog::
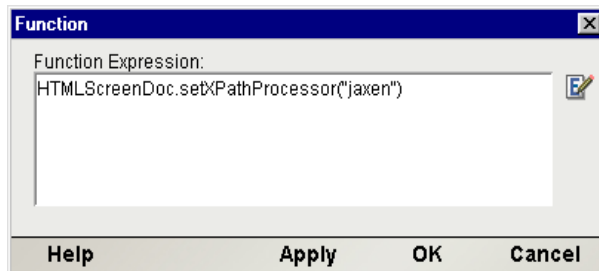
When you are doing load testing or performance tuning, you should try each XPath processor to see which is better for your particular application.

NOTE: If your application has bugs when running under one XPath processor but not with the other, it is most likely because your application depends on DOM nodes being returned (from XPath expressions) in a given order. Always remember, when using any XPath expression that returns a node list, that the ordering of nodes in the node list is not predictable. This is standard XPath behavior across all processors. Your application should not depend on node objects being returned in any particular order.

JavaScript versus ECMAScript

ECMAScript is the standards-body-blessed "core language" underlying JavaScript. (Consult ECMA Standard Number 262, published by Ecma International, formerly the European Computer Manufacturers Association. For details, go to **http://www.ecma-international.org/**.) JavaScript is a superset of ECMAScript: It is ECMAScript plus the various browser extensions objects and methods supported by the browser-makers, plus HTML DOM extensions. The HTML Connect "understands" the most common JavaScript browser extensions, including many Netscape and Mozilla objects, but not JScript-specific or Internet Explorer-specific methods.

User Agent Info

HTTP clients can identify themselves to a web server by a User-Agent request-header field. The server can use this information in a variety of ways. (For instance, it can tailor outgoing content to suit a particular browser type, perhaps redirecting a Mozilla user to pages containing Mozilla-specific markup, or Internet Explorer users to pages containing IE-specific scripts or markup.) Since user-agent info is also available as a JavaScript property, web-page authors often use JavaScript to learn which kind of browser the page is running in.

Composer's HTML Connect uses the following User-Agent string:

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2) Gecko/20030208
Netscape/7.02
```

(This represents one continuous string. Linewrapping is unintentional.)

Web page scripts can inspect this property by means of:

```
browserType = navigator.userAgent;
```

The "navigator" object is a standard browser-JavaScript construct, supported by Composer.

In essence, any components you build using the HTML Connect for Composer will "appear to be" a Mozilla browser to any servers or scripts that check the user-agent property.

Handling Errors and Messages

In the event a Get operation is not successful, the HTTP status code will be available via ECMAScript in the `HTMLStatusCode` property. Also, a property called `HTMLStatusMsg` will contain a string with information about the error condition. The contents of these properties can be captured in log messages using Log Actions. (See the Section on "The Log Action" of the *Composer User's Guide* for more information about Log Actions.)

NOTE: Refer to Appendix C for a complete listing of HTTP Status Codes.

If a script inside an HTML page that uses JavaScript is not understood by Composer, you may see this message in the logs or Output tab:

```
java.lang.RuntimeException: com.novell.mozilla.javascript.PropertyException:
Constructor for "TypeError" not found.
```

(Linewrap is unintentional.)

Bear in mind that some web pages assume Internet Explorer compatability or use IE JavaScript/JScript objects that are not supported by Mozilla clients. (For all intents, Composer's HTML Connect is a Mozilla client. See "User Agent Info" above.) This can be a possible source of trouble. If your component is "scraping" a page intended for Microsoft IE clients, and or if you are going against a page that uses MSIE-dependent scripting constructs, your component may not work as expected. There is little you can do at this point, unless the site in question offers non-scripted pages for use by clients that lack script support or that have JavaScript turned off.

A **Digital Certificates**

HTML Connect supports ninety-eight (98) digital certificates in the jar file *agrootca.jar.* The following table lists the certificates. If you need to add a certificate to the jar file, use the procedure described following the table.

Authorities Certification Supported by exteNd
ABA.ECOM Root CA
Baltimore EZ by DST
Belgacom E-Trust Primary CA
CA Data
Certiposte Classe A Personne
Certipose Serveur
Certisign - Authoridade Certificadora - AC2
Certisign - Authoridade Certificadora - AC4
Certisign - Authoridade Certificadora - AC1S
Certisign - Authoridade Certificadora - AC3S
Class 1 Primary CA
Class 1 Public Primary Certification Authority - G2
Class 1 Public Primary Certification Authority - G2_2
Class 1 Public Primary Certification Authority
Class 1 Public Primary Certification Authority_2
Class 2Primary CA
Class 2 Public Primary Certification Authority - G2
Class 2 Public Primary Certification Authority - G2_2
Class 2 Public Primary Certification Authority - G2
Class 2 Public Primary Certification Authority- G2_2
Class 3 Primary CA
Class 3 Public Primary Certification Authority - G2
Class 3 Public Primary Certification Authority - G2_2
Class 3 Public Primary Certification Authority
Class 3 Public Primary Certification Authority_2
Class 3P Primary CA

Authorities Certification Supported by exteNd
Class 3TS Primary CA
Class 4 Public Primary Certificate Authority - G2
Class 4 Public Primary Certificate Authority - G2_2
Commercial Certification Authority
Deutsche Telekom Root CA 1
Deutsche Telekom Root CA 2
DST (ANX Network) CA
DST (NRF) RootCA
DST (UPS) RootCA
DST RootCA X1
DST RootCA X2
DST-Entrust GTI CA
DSTCA E1
DSTCA E2
Entrust.net Secure Server Certification Authority
Equifax Secure eBusiness CA
Equifax Secure Global eBusiness CA
FESTE, Public Notary Certs
FESTE, Verified Certs
First Data Digital Certificates Inc. Certification Authority
FNMT Clase 2 CA
getcacert
GlobalSign Root CA
GTE CyberTrust Global Root
GTE CyberTrust Root
GTE CyberTrust Root_2
IPS SERVIDORES
KeyWitness 2048 Root
Microsoft Authenticode(tm) Root Authority
Microsoft Corporation
Microsoft Root Authority
NetLock Expressz (Class C) Tanusitvanykiado
NetLock Kozjegyzoi (Class A) Tanusitvanykiado
NetLock Uzleti (Class B) Tanusitvanykiado
PTT Post Root CA

Authorities Certification Supported by exteNd
Secure Server Certification Authority
SecureNet CA Class A
SecureNet CA Class B
SecureNet CA Root
SecureNet CA SGC Root
SecureSign RootCA1
SecureSign RootCA2
SecureSign RootCA3
SERVICIOS DE CERTIFICACION - A.N.C.
SIA Secure Client CA
SIA Secure Server CA
Swisskey Root CA
TC TrustCenter Class 1 CA
TC TrustCenter Class 2 CA
TC TrustCenter Class 3 CA
TC TrustCenter Class 4 CA
TC TrustCenter TimeStamping CA
Thawte Personal Basic CA
Thawte Personal Freemail CA
Thawte Personal Premium CA
Thawte Premium Server CA
Thawte Server CA
Thawte Timestamping CA
UTN - STATCorp SGC
UTN - USERFirst-Client Authentication and Email
UTN - USERFirst-Hardware
UTN - USERFirst-Network Application
UTN - USERFirst-Object
ValiCert Class 1 Policy Validation Authority
ValiCert Class 2 Policy Validation Authority
ValiCert Class 3 Policy Validation Authority
VeriSign Commercial Software Publishers CA
VeriSign Commercial Software Publishers CA_2
VeriSign IndividualSoftware Publishers CA
VeriSign IndividualSoftware Publishers CA_2

Authorities Certification Supported by exteNd
VeriSign, Inc.
Xcert EZ by DST

➢ **To add an additional trusted certificate authority to exteNd:**

1   In a WinZip utility program, open the **Designer/lib/xcroota.jar** file.

2   Click Add, browse from the directory you want to add the file.

3   Add the file to the rest of the files listed within the **xcroota.jar** file. Close the WinZip program.

4   Restart your server.

B Testing

Environmental Differences between Animation Testing and Deployment Testing

There are significant environmental differences between Animation testing in Composer and Deployment testing. Both types of testing are needed to adequately verify the components and services you build. The differences are detailed in the table below.

Issue	Testing in Composer	Deployment Testing
Images in HTML pages	Rendered in Native Environment Pane.	No rendering.
Error handling	Error messages appear in Native Environment Pane.	HTTP status codes can be captured via Log Actions.
Project Variables for: * Log File Paths * DTD URIs * XSL URIs * Send Mail Server * XML Inter- change URIs	Usually point to locations on local machine (but could be on Servers or Web).	Should point to locations on production Servers and Web.
Testing Tools	In addition to Log actions, you can use dialog boxes, via ECMAScript *alert()* function, to display runtime values.	No dialog boxes can be used.

C HTTP Status Codes

The HTTP Status Code is a 3-digit integer result code summarizing the outcome of a particular HTTP request. These codes are summarized below. The first digit of the Status Code defines the class of response. The last two digits have no particular "category" meaning.

There are five classes of Status Code based on the first digit of the code:

1xx	Informational	Request received, continuing process
2xx	Success	The request was successfully received, understood, and accepted
3xx	Redirection	Further action must be taken to complete the request
4xx	Client Error	The request contains bad syntax or cannot be fulfilled
5xx	Server Error	The server failed to fulfill an apparently valid request

Detailed Code Semantics

100 Continue

101 Switching Protocols

200 OK

201 Created

202 Accepted

203 Non-Authoritative Information

204 No Content

205 Reset Content

206 Partial Content

300 Multiple Choices

301 Moved Permanently

302 Found

303 See Other

304 Not Modified

305 Use Proxy

307 Temporary Redirect

400 Bad Request

401 Unauthorized

402 Payment Required

403 Forbidden

404 Not Found

405 Method Not Allowed

406 Not Acceptable

407 Proxy Authentication Required

408 Request Time-out

409 Conflict

410 Gone

411 Length Required

412 Precondition Failed

413 Request Entity Too Large

414 Request-URI Too Large

415 Unsupported Media Type

416 Requested range not satisfiable

417 Expectation Failed

500 Internal Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

504 Gateway Timeout

505 HTTP Version not supported

extension-code = 3DIGIT Reason-Phrase = *<TEXT, excluding CR, LF>

NOTE: HTTP status codes are extensible. HTTP applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable.

D Actions Created When Form Field Values Are Modified Interactively

When you are in Record mode and you interact with form fields inside the web page currently displayed in the Native Environment Pane, appropriate actions are automatically added to the Action Model in real time. For example, when you enter text into a text field in an HTML page, a Map Action is automatically generated. The following table shows which types of Actions are autogenerated in the Action Model when various field types are manipulated.

Field Type and User Action	Action(s) Created in the Action Model
User enters (or drags) text into a text field	Map Action: ```MAP string TO HTMLScreenDoc.XPath(path)```
User checks a check box	Map Action. Example: ```MAP true TO HTMLScreenDoc.XPath("//input[@name = 'safe' and @value = 'on']").createXPath("/@checked")```
User unchecks a check box	Function Action. Example: ```CALL removeCheckedAttr(HTMLScreenDoc.XPath("//input[@name = 'safe']"));```
User selects a radio button	Map Action. Example: ```HTMLScreenDoc.XPath("//input[@name = 'safe' and @value = 'on']").createXPath("/@checked")```
User deselects a radio button	Function Action. Example: ```CALL removeCheckedAttr(HTMLScreenDoc.XPath("//input[@name = 'safe']"));```
User selects an item from a dropdown list	Function Action. Example: ```updateSelectedAttr(HTMLScreenDoc.XPath("/descendant::select[@name = 'num']/option"), '20 ');```
User toggles an item in a list box	Function Action: ```CALL toggleListSelectedAttr(DOMelement, string, boolean)``` NOTE: See next Appendix for source code and usage of this and related custom ECMAScript function.

User clicks Submit button	Map Action followed by HTTP Post (typically). Example:
	
	```CALL HTMLScreenDoc.XPath("//input[@name = 'Mode' and @value = 'SEARCH']").createXPath("/@checked")```
	```HTTP/HTML Post From "http://www.xyz.com/search.php"```

E Internal Scripts used by Recorded Function Actions

```
// These ECMAScript functions are invoked from Function Actions created
// in the HTML Connect when a user makes certain selections on HTML FORM
// based visual controls.
/*********************************************************/
// Functionname: removeSelectedAttr(nodelist)
// Description:  removes 'selected' attribute from a control
// nodelist:     is required, the list of nodes to remove attribute from
// Returns:      void
// Note:         Uses DOM Element methods getAttribute()
//               and removeAttribute()
/*********************************************************/
function removeSelectedAttr(nodelist)
{
   for(var elem in nodelist)
   {
      var attr = elem.getAttribute("selected");
      if(attr != null)
         elem.removeAttribute("selected");
   }
}


/*********************************************************/
// Functionname: updateSelectedAttr(nodelist, asRExpr)
// Description:  Method to update list selection for a COMBO BOX. A Combo Box can
//               have one value selected. This method compares the passed in
//               expression and creates selected attr for the one element whose
//               attribute matches the passed in param and removes selected attr
//               for the rest of the items, in case they have selected attr
present
// nodelist:     is required, is the list of option nodes
// asRExpr:      is required, is the right hand side expression
// Returns:      void
// Note:         Uses DOM Element methods getAttribute() and removeAttribute()
/***************************************************************/
function updateSelectedAttr(nodelist, asRExpr)
{
   for(var elem in nodelist)
   {
      // get the node value. Call back exteNd
      var lVal1 =
Packages.com.sssw.b2b.rt.GNVXMLDocument.getNodeStringValue(elem);

      // trim leading and trailing blanks on both side of the expressions
      // before comparison
      var lVal = new java.lang.String(lVal1);
      lVal = lVal.trim();
      var rVal = new java.lang.String(asRExpr);
      var rVal = rVal.trim();
      if(lVal == rVal)
         elem.setAttribute("selected", "selected");
      else
      {
```

```
                    var attr = elem.getAttribute("selected");
                    if(attr != null)
                        elem.removeAttribute("selected");
                }
            }
        }

        /***********************************************************/
        // Functionname: removeCheckedAttr(nodelist)
        // Description:  Method to remove 'checked' attribute for RADIO BUTTONS,
        //               CHECKBOXES, PUSH BUTTONS etc.
        // nodelist:     (required) list of nodes for which 'checked' attr needs
        //               to be turned off.
        // Returns:      void
        // Note:         Uses DOM Element methods getAttribute()
        //                and removeAttribute()
        /***********************************************************/
        function removeCheckedAttr(nodelist)
        {
            for(var elem in nodelist)
            {
                var attr = elem.getAttribute("checked");
                if(attr != null)
                    elem.removeAttribute("checked");
            }
        }

        ***************************************************/
        // Functionname: toggleListSelectedAttr(elem, asValue, abAddAttr)
        // Description:  Method to create selected attribute
        //               for a MULTI-SELECTION LIST BOX in case the boolean
        //               passed in param abAddAttr is true, and
        //               remove selected attr in case abAddAttr is false.
        // elem:         DOM element for which 'selected' attr is either to be
        //               created or removed.
        // asValue:      value to be matched for the 'option' desendant element.
        // abAddAttr:    boolean. When true, creates 'selected' attr
        //              for the 'option'
        //               element  for which the value matched with
        //               asValue and removes
        //               'selected' attr in the event the flag is false.
        // Returns:      void
        // Note:         Uses DOM Element methods getAttribute()
        //                and removeAttribute()
        /***********************************************************/
        function toggleListSelectedAttr(elem, asValue, abAddAttr)
        {
            // find all descendant elements by the name 'option'
            var lOptionList = elem.getElementsByTagName("option");
            for(var child in lOptionList)
            {
                // get value attr for the child
                var lsChildAttrVal = child.getAttribute("value");
                if(lsChildAttrVal == asValue)
                {
                    // for the child whose value matches passed in value,
                    // create attr in case abAddAttr is true or remove otherwise
                    if(abAddAttr == true)
                    {
                        child.setAttribute("selected", "selected");
                    }
                    else
                    {
                        child.removeAttribute("selected");
                    }
                    break;
```

```
                }
            }
        }
```

F HTML Glossary

Cookie   A text file or string that stores state information. Some cookies are scoped to the HTTP session only (no longer existing once the client application shuts down) while others are *persistent* by virtue of being written to a storage device.

Digital Certificate   Enables secure communication with a Web server.

Field   A unit of data contained in a form. A field may be a label to display on the screen, an item of data, or an interactive widget of some kind. Some fields may be hidden (and thus have no visual interface.) Each field has its own attributes that determine how it is displayed and if the area can be modified.

Frames   Some HTML pages are displayed in discrete panes with draggable dividers. These panes are called *frames*. The FRAMESET tag is used to create a group of frames. Each frame has its own FRAME tag. The source code for an HTML page that contains frames does not contain the detailed HTML code for the individual frames; instead, the SRC attribute of the FRAME element directs the browser to the appropriate URI for the frame.

HTML   Hyper Text Markup Language

HTTP   HyperText Transfer Protocol.

HTTP GET   A method that gets the file at a specified URI using URIencoded form data appended to the target URI.

HTTP POST   A method that posts data to a specified URI (which usually points to a CGI script on the server).

HTTPS   The "secure" version of HTTP.

Native Environment Pane   A pane in the HTML Component Editor that provides an emulation of an actual HTML terminal session.

Map Screen Action   A special non-editable action that indicates the location in an Action Model where a new data is received. Any actions intended to interact with this screen must be placed subordinate to the Map Action's Screen Actions line in the Action Model.

Redirection   An attempt to send the client to a new destination URI. Redirection may occur as a result of a Refresh directive in the <META> elements of an HTML page, a specific Refresh directive issued by the server (in the HTTP header of a response), or a script (typically JavaScript) inside the HTML page.

ScreenDoc   A special DOM that can be displayed in the HTML Component Editor representing the current HTML page as an XML document.

SSL   Secure Socket Layer Support

TIDY   A utility used to transform hard to read HTML into clearly layered markup text.

URI Uniform Resource Locator. The URI expresses the location of the web resource as well as the protocol that should be used to communicate thereto.

Web Browser A display window where the actual contents of a HTML document is displayed.

XHTML HTML meeting the minimum "well-formedness" requirements of the XML standard.

G Reserved Words

The following terms are reserved words in exteNd Composer for the HTML Connect and should be avoided in any user created labels or variable names.

- Get
- Post
- Put
- Redirect
- HTML
- ScreenDoc
- FrameSet
- theComponent

Index