AspDotNetStorefront
e-commerce by design

XmlPackage Specification v2.1
for AspDotNetStoreFront Version 6.0 or higher

Updated 4/2/2006

XmlPackage Overview & Operation

XmlPackages are a means by which developers can extend the functionality of AspDotNetStorefront without necessarily having to know asp.net and/or recompile the storefront. A knowledge of Xml, Xsl (Xslt), and SQL is required.

XmlPackages have two purposes:

1) Internal Packages: these are used internally by AspDotNetStorefront to provide efficient access to nested "entity" (e.g. category, section, library, etc) and "object" (e.g. product, document) structures in the database. The storefront will NOT function if these XmlPackages removed.

2) Display Rendering Packages: These XmlPackages are created by third party developers (anyone really) to allow generation of custom page HTML output for entity and/or object pages. An example is to create a custom category page display format for the storefront.

When Upgrading to AspDotNetStorefrot 6.1

When you upgrade to 6.1, we STRONGLY recommend that you remove your parser tokens from all of your XmlPackages and use the direct XsltExtension objects, documented in the manual. If you want to retain your pre 6.1 packages and use them on 6.1, you will have to add this to the xmlpackage file:

RequiresParser="true"

If you don't do this, the tokens will be out put in plain text (e.g. (!FormatCurrency value="1.00"!)...).

The syntax would be:

<package version="2.1" displayname="Flower Road" debug="true" RequiresParser="true">

Again we recommend for performance reasons that you rewrite your XmlPackages when upgrading to version 6.1 to take advantage of the XslExtension objects provided by the storefront.

Also, the package format used in pre-5.9 versions of the store (those with the .xslt extension) are no longer supported in version 6.1.

What is an XmlPackage

Conceptually, an XmlPackage is pretty simple. An XmlPackage takes an input data set (or SQL statement), combines it with system and customer parameters, and then produces output in either Xml for HTML format.

XmlPackages are named with the .xml.config extension (e.g. MyXmlPackage1.xml.config). They are named this way to prevent someone from "browsing" your XmlPackage file.

The steps by which an XmlPackage performs is like this:

1) Package is instantiated by Name (the filename, e.g. MyXmlPackage1.xml.config)
2) Package reads .xml.config file (we use Xml structure itself to define the specification of the XmlPackage, and we add .config as an extension to prevent anyone from viewing your XmlPackages stored on your web site with a browser, for additional security protection).
3) Code adds additional run-time parameters to the XmlPackage
4) Code adds additional system defined parameters to the XmlPackage
5) DataSets are built from the SQL statements in the XmlPackage
6) DataSets are converted into XmlDocument fragments and combined with the system data (also XmlDocument fragments) to form the Xml Data Document.
7) XmlDocument is Xsl "Transformed" into final Xml or HTML

The technologies of: Xml, Xsl, SQL, HTML, XHTML, etc are all required probably to write effective XmlPackages. There are many tutorials and good books on those technologies, so they are not discussed here, except where any unusual exceptions or requirements are imposed by the storefront.

Xml documents are CasS SEnSITivE!!! Also Xmldocuments must have various attributes and element values properly "Xml Encoded". There is and XSD schema in the /XmlPackages folder to help you create properly structured packages.

Internal XmlPackages

When used internally as a helper object, the XmlPackage typically produces an in-memory XmlDocument object, as a result of input SQL statements, and various customer, system, and user-defined parameters. Examples of these internal packages can be found by looking at files such as:

- EntityHelper/EntityMgr.xml.config
- etc…

This manual does not describe in further detail how the XmlPackages are internally used by the storefront for managing entities. You can refer to the EntityHelper.cs, XmlPackage2.cs, and HierarchicalTableMgr.cs classes in the common project, and the ShowEntityPage.cs class in the storefront project. Various consumer pages of these objects include the SiteMap.cs classes, and the SiteMapComponentArt.cs classes.

Display Rendering XmlPackages

When used by developers or customers, this type of XmlPackage outputs "HTML" to be rendered within a store page, typically rendering within the skin in the contents area.

We now discuss how users can add new XmlPackages to the storefront WITHOUT recompiling any code in the storefront. This is important to allow developers to add custom page formats for their clients. For example, a particular store client may want their category pages to be displayed in a certain manner, which is not one of the

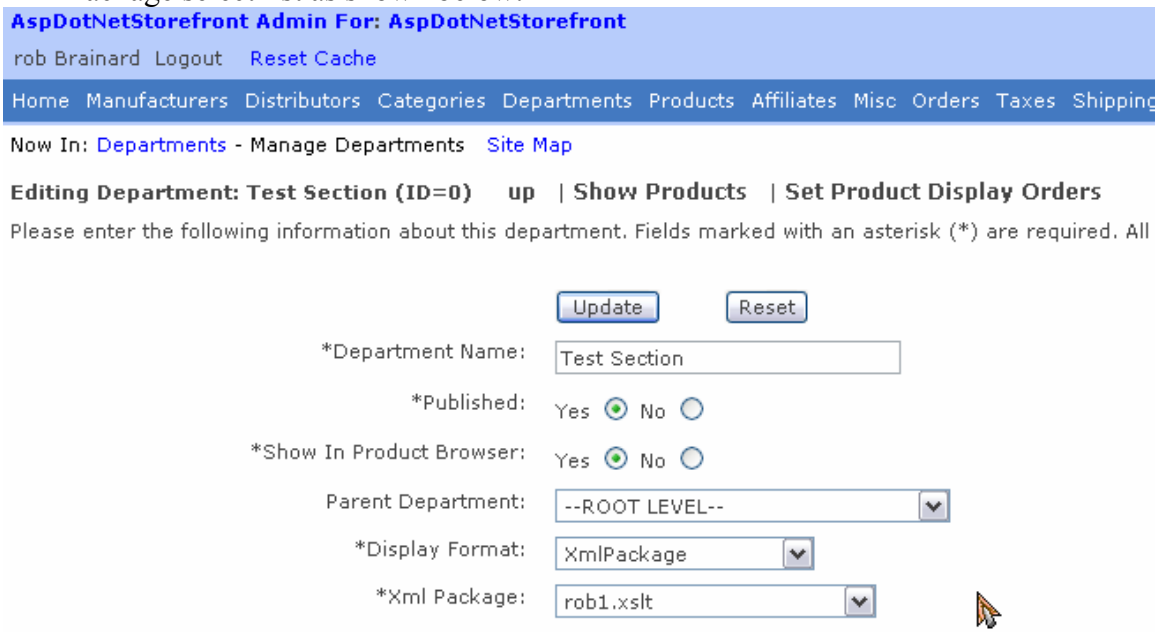
built-in provided formats of the software. The developer can then author an XmlPackage (in notepad if required), copy that XmlPackage to the server, and tell the storefront to use that XmlPackage to render the specified category(ies) via the Admin site.

Installing XmlPackages

XmlPackages are installed by placing them in the skins/skin_(!SKINID!)/XmlPackages folder in the web site. XmlPackages are stored by skin, because different skin layouts may require different manifestations of the package. Also, the admin/skins/skin_1/XmlPackages contains the XmlPackages (typically Internal XmlPackages) that are needed by the Admin site to function properly.

When your XmlPackage is ready for use, just FTP the file, or copy the file into that directory.

Then, you can assign that XmlPackage to any category, section, manufacturer, library, product, or document in the storefront using the Admin site. When editing these pages in the Admin panel, you will now select the Display Format of "XmlPackage" in the select list, and then pick which XmlPackage file is to be used in the Xml Package select list as shown below:



When the display format of XmlPackage is chosen for your page, your XmlPackage is solely responsible for producing the output from that page's contents areas! This gives you COMPLETE control over what is rendered within the skin content area.

Invoking XmlPackages By Themselves

The new page "engine.aspx?xmlpackage=XXXXXX" is newly added to the storefront also. The static URL shorthand for this is e-XXXXXX.aspx.

This page allows you to invoke any XmlPackage specified in the query string, and have it's contents rendered in the page contents area. Note that not all XmlPackages can be executed in this manner, as many XmlPackages DEPEND on system, page, or user information (e.g. query string categoryid= string, etc).

However, much like topic pages produce standalone page output, XmlPackages "can" be used in the engine.aspx page to produce standalone XmlPackage driven page output. This basically extends the topic

concept to a programmatic level, allowing developers to add dynamic content pages to the storefront without ever touching the source code.

XmlPackages also allow you to add totally new data driven pages to your site. Similar to "topics" which are static HTML block pages, there is a new page in the storefront called engine.aspx which can be used to invoke XmlPackages. The syntax is:

engine.aspx?xmlpackage=nameofpackage

the URL shorthand is:

e-nameofpackage.aspx

(Example: e-helloworld.aspx)

This new engine page now allows you to do things using dynamic data to add totally new pages in your store. For example, suppose you want to have a page which lists the top 5 most heavily discounted products in your db. You can now write an XmlPackage to do that, and link to it from your skin. Suppose you need a page which lists the most 25 recently added products in the store, you can write an XmlPackage to do that. And with both packages, you have full control over the data that is being used to generate the page, and the display format used to render the page results. All this is possible by writing SQL queries and XSL transforms files and combining them into an XML file according to our simple format. No change to the asp.net storefront code is required. To deploy a new XmlPackage page to your site, all you have to do is FTP the XmlPackage file to the /XmlPackages directory (or skins/skin_(!SKINID!)/XmlPackages directory) of your web server and invoke it (presumably after you have tested and debugged it on your development server).

Whe executing XmlPackages in this way you may want to override the page title, keywords, description, etc. There are five elements that you can include in the package to set these values. They are included in the SearchEngine setting node and are named: `SectionTitle`, `SETitle`, `SEKeywords`, `SEDescription`, and `SENoScript`. Each of the elements can contain either an xpath statmement that locates the element within the Xml Data Document that contains the data or a Xslt stylesheet to transform multiple nodes in to one value to be placed in the output. The XPath statement must return only one node and the stylesheet should be designed to work with the full Xml Data Document.

Debugging XmlPackages

You can set AppConfig:Xml.DumpTransform=true, and the XmlPackage engine will write intermediate .xml files into the /image directory (they are put here because normally the site already has write permissions to that folder). The xml files will be appropriately named based on the name of the xml package, and also intermediate stage files will be written out.

The above procedure will cause debug information for every package used on the page to be display. If you only want to debug a specific package you can add the debug=”true” attribute to the paclage element. This will casue bebug information to be display regardless of the setting the AppConfig:Xml.DumpTransform.

To debug your xsl transforms, you can use your favorite Xsl Debug tool to take the intermediate .xml data file dumped out, and apply your transform directly to it. Also, Visual Studio 2005 will debug transforms in-line. In order to do this you will need to put a break point in the XmlPackage2.TransformString method before the m_Transform.Transform method is called. When the code reaches your breakpoint, chose the File|Open menu item and open the transform file in the images folder, the file will be named fullpackagename_store.runtime.xsl (e.g. product.SimpleProduct.xml.config_store.runtime.xsl). After you have the file opened you can set breakpoints in the transform and view local variables (i.e. param or variable tags in the transform) as you step

through the transform. Again you must turn on debugging for the package using the package element's debug attribute (debug="true") or by setting the AppConfig parameter Xml.DumpTransform to true.

XmlPackage Structure

XmlPackages can contain:

1) SQL Queries to be executed
2) Web Queries
3) Xsl transforms
4) Search Engine Setting definitions
5) Post Processing Queries
6) Set Cookie instructions

An XSD Schema (XmlPackages/xmlpackage.xsd) is provided to describe the structure of the package. You can use this schema to validate the structure of your package (this does not validate the logic or structure any of the Xslt stylesheets or sql queries).

SQL Queries

The Query element is defined as:

<query name="Sections" rowElementName="Section" runif="*paramname*">

The name attribute is used for the name of the child of the root element in the XML Data Document for this query. The rowElementName is the element name for each row returned by the query. Elements for each field are created using the exact spelling of the field or field alias in the query. Remember, all names are case sensitive. So, to reference the Name field from a query that has the above definition you would use the following path /root/Sections/Section/Name. The runif attribute can be used to optionally run the query. The value should be either a querystring/form/cookie param or an appconfig param. If the specified querystring/form/cookie param or an appconfig param doesn't exist or is an empty the query will not be executed. This could be used for a page where the query should not be run until the page is submitted with a form field.

Each query has a <sql> element that can contain any valid SQL statement. The SQL statements are executed in .NET using the SqlClient data provider and are executed as parameterized queries.

Query Parameters:
There are two types of query parameters that can be defined, the queryparam element and querystringreplace element.

A queryparam element defines a parameter that is evaluated as SQL execution time and has the format @paramname. For each queryparam element ther must be a @paramname variable in the SQL statement. (e.g. select * From product where productid = @productid). The queryparam element has the following attributes (all of which are required):

- **paramname** – the name of the param as it is in the SQL statement.
- **paramtype** – defines where the parameter value is obtained, valid values are request, appconfig, and runtime
    - request param values are retrieved from the Web request object (i.e. a Querystring, Form, Cookie value)

    - appconfig params are retireve from the AppConfig table
    - runtime params are retieved from an internal runtime parameters table
- **requestparamname** – this is the name of the request, appconfig, or runtime field that is used for the value to be substituted in the query. It must match exactly or the item will not be found or it may find a different parameter altogether
- **sqlDataType** – defines the .Net SqlDataType for this parameter. The allowed types can be view in the XmlPackages/xmlpackage.xsd file, they are defined in the elment definition <xsd:simpleType name="SqlDataType">.
- **defvalue** – the default value to use in case the value is not found in the specified request, appconfig, or runtime collection
- **validationpattern** – a regular expression that can be used to filter the data so that invalid ranges of values are not sent to the query.

The second type of query parameter is the querystringreplace parameter. It is used to may the SQL string dynamic in terms of it table names, field name, where clause fields, and order by clause fields. **It is not intended to be used for filtering parameters in the WHERE clause (for security reasons).** This element has the following attributes (all of which are required):

- **replaceTag** – a user defined string embedded in the SQL statement that will be replaced at run time.
- **replacetype** - defines where the string replacement value is obtained, valid values are request, appconfig, and runtime and behave the same as those defined above in the queryparam element.
- **replaceparamname** - this is the name of the request, appconfig, or runtime field that is used for the value to be substituted for the replaceTag string in the query. It must match exactly or the item will not be found or it may find a different parameter altogether.
- **defvalue** - the default value to use in case the value is not found in the specified request, appconfig, or runtime collection
- **validationpattern** – a regular expression that can be used to filter the data so that invalid ranges of values are not sent to the query.

Here is an example query element:

```
<query name="Entities" rowElementName="Entity">
  <sql>
    <![CDATA[
      select Name,Description from {EntityName} with (NOLOCK) where {EntityName}ID=@EntityID
    ]]>
  </sql>
  <querystringreplace replaceTag="{EntityName}"
                      replacetype="runtime"
                      replaceparamname="EntityName"
                      defvalue=""
                      validationpattern="(category)|(section)|(affiliate)|(manufacturer)|(distributor)|(library)" />
  <queryparam paramname="@EntityID"
              paramtype="runtime"
              requestparamname="EntityID"
              sqlDataType="int"
              defvalue="0"
              validationpattern="" />
</query>
```

It would produce a document fragment like this

```
<Entities>
	<Entity>
		<Name>Test Entity1</Name>
		<Description>Test Description</Description>
	</Entity>
	<Entity>
		<Name>Test Entity2</Name>
		<Description>Test Description</Description>
	</Entity>
	…
</Entities>
```

The query element can also contain an XSL transform element named querytransform. This element can contain an XSL transform that can be used to shape the output XML for this query differently than it comes from the database and before it is added to the final XML Data Document. It is optional but there can be no more than one transform for the query. The output of this transform must be a valid XML document fragment or the entire package will fail.

Web Queries

Web queries allow you to get data from an external data source via URL. The URL can return xml or text data (as specified by the RetType attribute). The name attribute is used for the node name in the Xml Data Document under which all returned content is inserted. The url element contains the url of the web document, including any querystring parameters, to get the data from. The URL should be fully formed, e.g. http://www.somesite.com/xmldoc.aspx?param1=123. The querystring replace element is used much the same as in sql queries. It describes a tag in the URL string that can be replaced by some value from a request (querystring, form, cookie, or server) variable, a runtime variable or an appconfig parameter. Finally, the query can contain an XSL transform. If a transform is specified, the returned data will be transformed and the results of the transform are added to the XML Data Document instead of the raw results from the URL. Tranforms are only run when the RetType attribute is "xml". Again you can refer to the schema for a more technical description of the webquery element. A RetType of text causes the return data to be put in a CDATA element unmodified from how it was received.

Example:

```
<webquery name="WebData1" RetType="xml">
	<url>http://www.somesite.com/xmldatafeed.aspx?param1={param1}</url>
	<querystringreplace replaceTag="{param1}" replacetype="request"
		replaceparamname="productid" defvalue="0"
		validationpattern="^\d{1,10}$"/>
</webquery>
```

System Defined DataSets

In addition to your defined SQL queries, the storefront automatically adds the following DataSets to the XmlPackage before execution. This means that all of these datasets are available to your Xsl Transform via intermediate Xml Document:

System Data Set:

```
<System>
  <IsAdminSite>False</IsAdminSite>
  <IsAdminSiteInt>0</IsAdminSiteInt>
  <PublishedOnly>1</PublishedOnly>
  <CustomerID>0</CustomerID>
  <CustomerLevelID>0</CustomerLevelID>
  <CustomerLevelName />
  <CustomerFirstName />
  <CustomerLastName />
  <CustomerFullName />
  <SubscriptionExpiresOn />
  <CustomerRoles />
  <IsAdminUser>false</IsAdminUser>
  <IsSuperUser>false</IsSuperUser>
  <LocaleSetting>en-US</LocaleSetting>
  <WebConfigLocaleSetting>en-US</WebConfigLocaleSetting>
  <SqlServerLocaleSetting>en-US</SqlServerLocaleSetting>
  <Date>11/30/2005</Date>
  <Time>1:28 AM</Time>
  <SkinID>1</SkinID>
  <AffiliateID>0</AffiliateID>
  <IPAddress>192.168.0.40</IPAddress>
  <QueryString>SectionID=1&amp;amp;SEName=test-section</QueryString>
  <UseStaticLinks>false</UseStaticLinks>
  <PageName>showsection.aspx</PageName>
  <FullPageName>/version60/showsection.aspx</FullPageName>
  <XmlPackageName>aspdnsf.xml.config</XmlPackageName>
  <StoreUrl>http://dotnetstorefront6/version60/</StoreUrl>
</System>
```

Runtime DataSet (a bit redundant with the runtime params, but still necessary and helpful sometimes in your Xsl transform):

```
<Runtime>
  <AffiliateID>0</AffiliateID>
  <CustomerID>0</CustomerID>
  <EntityName>Section</EntityName>
  <IsAdminUser>False</IsAdminUser>
  <SubscriptionExpiresOn />
  <UseStaticLinks>False</UseStaticLinks>
  <Date>11/30/2005</Date>
  <CustomerLevelName />
  <StoreUrl>http://dotnetstorefront6/version60/</StoreUrl>
  <Time>1:28 AM</Time>
  <EntityID>1</EntityID>
  <SkinID>1</SkinID>
```

<IsAdminSite>False</IsAdminSite>
<QueryString>SectionID=1&SEName=test-section</QueryString>
<CustomerLevelID>0</CustomerLevelID>
<CustomerFullName />
<CustomerFirstName />
<XmlPackageName>aspdnsf.xml.config</XmlPackageName>
<IsSuperUser>False</IsSuperUser>
<PublishedOnly>1</PublishedOnly>
<IsAdminSiteInt>0</IsAdminSiteInt>
<IPAddress>192.168.0.40</IPAddress>
<CustomerRoles />
<PageName>showsection.aspx</PageName>
<CustomerLastName />
<WebConfigLocaleSetting>en-US</WebConfigLocaleSetting>
<FullPageName>/version60/showsection.aspx</FullPageName>
<LocaleSetting>en-US</LocaleSetting>
<SqlServerLocaleSetting>en-US</SqlServerLocaleSetting>
<StoreUrl>http://dotnetstorefront6/version60/</StoreUrl>
</Runtime>

QueryString data set (whatever was on page URL invocation):

<QueryString>
<SectionID>1</SectionID>
<SEName>test-section</SEName>
</QueryString>

Form Data Set (whatever was on page FORM post):
<Form />

Session State (helpful with customer data and/or user defined session info):

<Session>
<CustomerID>58640</CustomerID>
<CustomerGUID>559ca809-884b-4072-b612-ac235c9ac743</CustomerGUID>
<ViewState>System.Web.UI.Triplet</ViewState>
</Session>

Server Variables Info (not used too often, but may be required in some situations):

<ServerVariables>
<HTTP_HOST>localhost</HTTP_HOST>
<HTTP_USER_AGENT>Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)</HTTP_USER_AGENT>
<AUTH_TYPE>Forms</AUTH_TYPE>
<AUTH_USER>559ca809-884b-4072-b612-ac235c9ac743</AUTH_USER>
<AUTH_PASSWORD />
<HTTPS>off</HTTPS>
<LOCAL_ADDR>127.0.0.1</LOCAL_ADDR>
<PATH_INFO>/aspdotnetstorefront/showsection.aspx</PATH_INFO>
<PATH_TRANSLATED>c:\websites\aspdotnetstorefront\showsection.aspx</PATH_TRANSLATED>

```
    <SCRIPT_NAME>/aspdotnetstorefront/showsection.aspx</SCRIPT_NAME>
    <SERVER_NAME>localhost</SERVER_NAME>
    <SERVER_PORT_SECURE>0</SERVER_PORT_SECURE>
   </ServerVariables>
```

These data sets are ALL available to your Xsl transform code, along with ALL the params you have defined, AND the runtime params added by the system.

XSL Transform

The output Package is obtained by applying an XSL transform to the resulting XmlDoc. The XmlDoc contains the SQL data, web data, and system data. This is done using the transform in the <PackageTransform> element. An XmlPackage doesn't require any query elements to run but the PackageTransform is required and must contain a valid XSLT stylehseet. As stated previously The XmlPackage uses .NET extension objects to implement high level logic not available in normal XSL. For this to work it is required that a new namespace be added to the XSL stylesheet. The xsl:stylesheet element must have the attribute xmlns:aspdnsf="urn:aspdnsf" here's the full stylesheet element:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:aspdnsf="urn:aspdnsf">
```

Although not required it is a good idea to use an output element as well and set the method attribute to html. Here's a sample

```
<xsl:output method="html" omit-xml-declaration="yes" />
```

After that all you need to do is add templates to transform the XML DataDocument into html.

Search Engine Settings

In some instances you may want to control the page title, keywords meta tag, and description meta tag, SectionTitle, and No Script sections. These items can be set by adding the SearchEngineSetting node to the package and including 1 or all of the following child nodes: SectionTitle, SETitle, SEKeywords, SEDescription, SENoScript. Each node can contain either an XPath statement, an Xslt stylesheet, or static text and you must specify which type it contains in the node's actionType attribute (either "xpath", "transform", or "text"). The XPath statement or the stylesheet is executed against the Xml Data Document. If the node contains an XPath statement, it must return a single node (a nodeset will cause a runtime error). This is for when you just want to pick the value of one element from the Xml Data Document. When a stylesheet is specified, you can combine as much information from the Xml Data Document as you need. Here is an example of specifying some of the SE settings:

```
   <SearchEngineSettings>
     <SETitle actionType="transform">
         <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:aspdnsf="urn:aspdnsf">
           <xsl:output method="html" omit-xml-declaration="yes" />

           <xsl:template match="/">
             <xsl:value-of select="concat(/root/Sections/Section/SEName, ' - ',
/root/Sections/Section/Description)" />
```

```
          </xsl:template>
        </xsl:stylesheet>
    </SETitle>
    <SEKeywords actionType="xpath">
        /root/Sections/Section/SEKeywords
    </SEKeywords>
    <SEDescription actionType="text">
        This is my static text to use for the search engine description meta tag
    </ SEDescription>
  </SearchEngineSettings>
```

When using a stylesheet you can use any of the extension functions described in the appendix A. Please refer to the schema (in the XmlPackages/xmlpackage.xsd file) for the full technical description of this node.

Post Processing

The XmlPackage can have post processing instructions. The post processing instructions include executing SQL queries (typically for insert or update), Web Queries, or setting cookie values. The actions take place after the XML DataDocument is created and before the transform is executed. So, after you have queried the database and/or retrieved data from the web you can execute commands to update your database, update your web data source, or set a user cookie value.

QueryAfter

Queryafter elements are used to define SQL queries that are executed after the XML DataDocument is built. This elements operates much like the <query> element except that the <querystringreplace> and <queryparam> elements can draw their values from the XML DataDocument (using an xpath statment). Typically you would use this to update data in your database after you gathered the data for the package transform. The <queryafter> element also contains a runif element that can be used to prevent the query from running. The runif element tests for the existence of the value or element you specify and if it does not exist or if it exists and is empty then the query is not executed.

Example:

```
<queryafter>
      <sql>
      <![CDATA[
            update customer set lastlogin = datetime where customerid = @custid
      ]]>
      </sql>
      <queryparam paramname="@ custid " paramtype="xpath"
                requestparamname="/root/System/CustomerID" sqlDataType="int" defvalue="0"
                validationpattern="" />
</queryafter>
```

Web Query After

Web query after is much the same as a webquery except that you can use xpath statements in the querystringreplace element to set it's value from a node in the XML DataDocument. Like the query after

element, web query after has a runif element that you can use to prevent the web query from being executed. If the parameter specified in the paramtype and paramsource attributes doesn't exist or is an empty string then the query is not executed.

Example:

```
<webqueryafter>
     <url>http://webservice.externalwebsite.com/xmldata.aspx?param1={param1}</url>
     <querystringreplace
          replaceTag="{param1}"
          replacetype="xpath"
          replaceparamname="/root/EntityHelpers/Category/Entity[1]/EntityID"
          defvalue=""
          validationpattern="" />
     <runif paramtype="appconfig" paramsource="MyCustomAppConfigParam" />
</webqueryafter>
```

AspDotNetStorefront Xsl Extensions

XmlPackages produce either:

1) Xml Output, or
2) HTML Output

When producing Xml Output, it is most likely your job to ensure that resulting Xml is what you intended, and complaint with Xml standards.

When producing HTML output, you can pretty much spit out whatever you want from the package, as long as you know how it will be used, and that it will be valid for that use.

However, when producing HTML, you may need to create blocks of HTML code which are non-trivial (e.g. an add to cart form with validation for a product).

Since some things would be extremely difficult to generate via XSLT. So, the HTML output from the XmlPackage can use custom XSL extension functions. These functions implement high-level logic that is not possible in XSLT alone (e.g. checking to see if an image file exists on the server before displaying it).

The functions are used in your packages like this:

<xsl:value-of select="aspdnsf:custfunction(arguments)" disable-ouput-escaping="yes">

The disable-output-escaping parameter is only required when the function's output is not valid XML. Also, XSL is case sensitive and that goes for the function names too. Also, note that all functions must be prefixed with aspdnsf:.

Supported functions are listed in **Appendix A**.

Putting XmlPackages in Skins

You can also embed an XmlPackage into your skin design directly, using the following syntax:

(!XmlPackage name=”abcd”!)

These packages will be invoked and replaced at run-time by the store skin parser. It is your responsibility to ensure that the package invoked is suitable for inclusion there, and produces the correct output.

Xsl Errors

While we want to assist you wherever possible, PLEASE do not call or e-mail our support group to help you diagnose YOUR Xslt errors. It is your job to know and understand Xsl, but if you find a bug in “our” code, let us know ☺

If you need paid XSL consulting, then by all means, please contact us.

Appendix A: AspDotNetStoreFront XSL Extension Functions

Function names are case sensitive
Boolean arguments are passed as 0 (false) or 1 (true).

Command	Description	Example
SkinID	Returns the active SkinID for the web page and current customer	<xsl:value-of select=”aspdnsf:SkinID()” >
CustomerID	Returns the currently logged in customer id, or 0 if anon customer with no customer record yet	<xsl:value-of select=”aspdnsf:CustomerID()” >
User_Name	Returns the currently logged in username (FirstName + “ “ + LastName), or empty string if anon user	<xsl:value-of select=”aspdnsf:User_Name()” >
User_Info	Returns a login form if the user is not logged in or displays the logged users name and a link their account page	<xsl:value-of select=”aspdnsf:User_Info()” >
User_Menu_Name	If not logged in the user will see the text in the skinbase.cs.7 string resource, otherwise it is the customer’s full name as entered on account.aspx page	<xsl:value-of select=”aspdnsf:User_Menu_Name()” >
Store_Version	Returns the text in the appconfig “StoreVersion” setting	<xsl:value-of select=”aspdnsf:Store_Version())” >
ManufacturerLink	Result is link to that manufacturer page in then site, using static links if required (e.g. m-1-mfg-se-name.aspx)	<xsl:value-of select=”aspdnsf:ManufacturerLink(ManufacturerID, SEName, TagInnerText)” disable-output-escaping=”yes”>

Argument name	Data Type	Description
ManufacturerID	Integer	The id of the manufacturer
SEName	String	Can be passed as an empty string but will be slower performing as it will attempt to read from the SEName from the database
IncludeATag		Returns an <a> tag with the href attribute set to the page url
TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)

CategoryLink	Returns a page name for the specified Category page in then site, using static links if required (e.g.c-1-mfg-se-name.aspx)	<xsl:value-of select="aspdnsf:CategoryLink (CategoryID, SEName, IncludeATag, TagInnerText)" disable-output-escaping="yes" />

Argument name	Data Type	Description
CategoryID	Integer	The id of the category
SEName	String	The SEName field from the db for this category, if known. If blank, the store will figure it out, but that takes an extra database query
IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)

SectionLink	Returns a page name for Section page in then site, using static links if required (e.g. s-1-mfg-se-name.aspx)	<xsl:value-of select="aspdnsf:SectionLink(SectionID, SEName, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	SectionID	Integer	The id of the section
	SEName	String	The SEName field from the db for this category, if known. If blank, the store will figure it out, but that takes an extra database query
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
LibraryLink			<xsl:value-of select="aspdnsf:LibraryLink(LibraryID, SEName, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	LibraryID	Integer	The id of the Library
	SEName	String	The SEName field from the db for this category, if known. If blank, the store will figure it out, but that takes an extra database query
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
ProductLink			<xsl:value-of select="aspdnsf:ProductLink(ProductID, SEName, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	The id of the Product
	SEName	String	The SEName field from the db for this category, if known. If blank, the store will figure it out, but that takes an extra database query
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
DocumentLink			<xsl:value-of select="aspdnsf:DocumentLink(DocumentID, SEName, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	DocumentID	Integer	The id of the Document
	SEName	String	The SEName field from the db for this category, if known. If blank, the store will figure it out, but that takes an extra database query
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
ProductandCategoryLink	Returns link to that product page (with Category breadcrumb navigation), using static links if required (e.g. pc-1-2-se-name.aspx)		<xsl:value-of select="aspdnsf:ProductandCategoryLink(ProductID, SEName, CategoryID, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	ProductID	Integer	Product ID
	SEName	String	Product Sename
	CategoryID	Integer	Category ID that the product is assigned to
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
ProductandSectionLink	Returns link to that product page (with Section breadcrumb navigation), using static links if required (e.g. p-1-2-se-name.aspx)		<xsl:value-of select="aspdnsf:ProductandSectionLink(ProductID, SEName, SectionID, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	Product ID
	SEName	String	Product Sename
	SectionID	Integer	Section ID that the product is assigned to
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
ProductandManufacturerLink	Returns link to that product page (with Manufacturer breadcrumb navigation), using static links if required (e.g. pm-1-2-se-name.aspx)		<xsl:value-of select="aspdnsf:ProductandManufacturerLink(ProductID, SEName, ManufacturerID, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	Product ID
	SEName	String	Product Sename
	ManufacturerID	Integer	Manufacturer ID that the product is assigned to
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
ProductProperName	Returns the “add to cart form” with javascript validation required to be able to let the site user add this product to the cart. This can include sizes, colors, inventory checking logic, etc..		<xsl:value-of select="aspdnsf:ProductProperName(ProductID, VariantID)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	VariantID	Integer	
DocumentandLibraryLink	Returns link to that document page (with Library breadcrumb navigation), using static links if required (e.g. pm-1-2-se-name.aspx)		<xsl:value-of select="aspdnsf:DocumentandLibraryLink(DocumentID, SEName, LibraryID, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	DocumentID	Integer	Document ID
	SEName	String	Product Sename
	LibraryID	Integer	LibraryID that the document is assigned to
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
EntityLink	Returns a link to the specified entity page		<xsl:value-of select="aspdnsf:EntityLink(EntityID, SEName, EntityName, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	EntityID	Integer	Category, Section, Manufacturer, or Document ID
	SEName	String	Product Sename
	EntityName	String	Entity Name (literally: Category, Section, Manufacturer, or Document)
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
ObjectLink	Returns link to that Object page in the site, using static links if required (e.g. p-1-se-name.aspx)		<xsl:value-of select="aspdnsf:ObjectLink(ObjectID, SEName, EntityName, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ObjectD	Integer	Category, Section, Manufacturer, or Document ID
	SEName	String	Product Sename
	EntityName	String	Entity Name (literally: Category, Section, Manufacturer, or Document)
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
ProductandEntityLink	Returns link to that product page (with Entity breadcrumb navigation), using static links if required (e.g. pm-1-2-se-name.aspx)		<xsl:value-of select="aspdnsf:(ProductID, SEName, EntityID, EntityName, IncludeATag, TagInnerText)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	Product ID
	SEName	String	Product SEName
	EntityID	Integer	Category, Section, Manufacturer, or Document ID
	EntityName	String	Literally: Category, Section, Manufacturer, or Document
	IncludeATag	Boolean (0/1)	Returns an <a> tag with the href attribute set to the page url
	TagInnerText	String	The text that goes inside the <a> tag, only used when the IncludeATag is true (set to empty sting otherwise)
Topic	Returns the specified topic by topic name or topic id. Only provide one of the arguments		<xsl:value-of select="aspdnsf:Topic(TopicName, TopicID)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	TopicName	String	
	TopicID	Integer	
AppConfig	Returns the specified AppConfig value		<xsl:value-of select="aspdnsf:AppConfig(AppConfigName)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	AppConfigName	String	
AppConfigBool	Returns the specified AppConfig value and a Boolean in lower case (i.e. “true’ or “false”)		<xsl:value-of select="aspdnsf:AppConfigBool(AppConfigName)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	AppConfigName	String	
StringResource	Returns the specified string resource value		<xsl:value-of select="aspdnsf:StringResource(StringResourceName)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	StringResourceName	String	String Resource name
LoginOutPrompt	Returns a login link or a logout link depending the users current login state		<xsl:value-of select="aspdnsf:LoginOutPrompt()" disable-output-escaping="yes"/>
SearchBox	Returns a form that contains an input control and submit button and posts to the search.aspx page		<xsl:value-of select="aspdnsf:SearchBox()" disable-output-escaping="yes"/>
HelpBox	Returns the contents of the “helpbox” topic.		<xsl:value-of select="aspdnsf:HelpBox()" disable-output-escaping="yes"/>
AddtoCartForm	Returns the “add to cart form” with javascript validation required to be able to let the site user add this product to the cart. This can include sizes, colors, inventory checking logic, etc..		<xsl:value-of select="aspdnsf:AddtoCartForm(ProductID, VariantID, ColorChangeProductImage)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	VariantID	Integer	
	ColorChangeProductImage	Boolean (0/1)	If the product has color options will selecting a color from the drop down cause the product image to change to the color selected. 1= yes, 0 = no
LookupImage	Returns the fully qualified <img src=”…”….> tag for the desired image. NoPicture or NoPictureIcon may be returned also in some cases. Note that this lookup honors lookup by sku, and image filename overrides set in the storefront. For medium images, the entire Product Image Gallery “html block” may also be returned, if that has been defined for the product. Also, if a medium size is requested, AND there is a large image available, the “show larger image” graphic and javascript code will be included.		<xsl:value-of select="aspdnsf:LookupImage(ID, EntityOrObjectname, DesiredSize, DesiredSize, IncludeATag)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	ID	Integer	Product, Category, Section, etc ID
	EntityOrObjectName	String	The name of the entity (“product”, “category”, “section”, etc.) for which you want the image
	DesiredSize	String	Literally: icon, medium, large, swatch
	IncliudeATag	Boolean (0/1)	Returns an <a> tag around the <img> tag with the href attribute set to the page url
LookupImage	Returns the fully qualified <img src=”…”….> tag for the desired image. NoPicture or NoPictureIcon may be returned also in some cases. This method differs form the one above in that the ImageFileNameOverride and SKU can be passed in and are not retirieved from the database. This method will not product a link to the large image when a medium image is requested. The SKU and ImageFileNameOverride arguments are optional and will cause the ID to be used as the filename.		<xsl:value-of select="aspdnsf:LookupImage(ID, EntityOrObjectname, DesiredSize, DesiredSize, IncludeATag)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ID	Integer	Product, Category, Section, etc ID
	EntityOrObjectName	String	The name of the entity (“product”, “category”, “section”, etc.) for which you want the image
	ImageFileNameOverride	String	The desired entity’s ImageFileNameOverride value, optional
	SKU	String	The desired entity’s SKU, optional
	DesiredSize	String	Literally: icon, medium, large, swatch
	IncliudeATag	Boolean (0/1)	Returns an <a> tag around the <img> tag with the href attribute set to the page url
LookupProductImage	Returns an <img> tag but specifically for a Product. The ImageFileNameOverride or SKUparameter will allow this method to perform much faster than the aove LookupImage function if you use ImageFileNameOverride or SKU on your product’s images. Either parameter can be and empty string if not used		<xsl:value-of select="aspdnsf:LookupProductImage(ProductID, ImageFileNameOverride, SKU, DesiredSize, IncliudeATag)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	ImageFileNameOverride	String	Product ImageFileNameOverride value or empty string is not used
	SKU	String	Product SKU value or empty string is not used
	DesiredSize	String	Literally: icon, medium, large, swatch
	IncliudeATag	Boolean (0/1)	Returns an <a> tag around the <img> tag with the href attribute set to the page url
LookupVariantImage	Returns an <img> tag but specifically for a Product Variant. The ImageFileNameOverride or SKUparameter will allow this method to perform much faster than the aove LookupImage function if you use ImageFileNameOverride or SKU on your product's images. Either parameter can be and empty string if not used		<xsl:value-of select="aspdnsf:LookupVariantImage(ProductID, VariantID, ImageFileNameOverride, SKU, DesiredSize, IncliudeATag)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	VariantID	Integer	
	ImageFileNameOverride	String	Product ImageFileNameOverride value or empty string is not used
	SKU	String	Product SKU value or empty string is not used
	DesiredSize	String	Literally: icon, medium, large
	IncliudeATag	Boolean (0/1)	Returns an <a> tag around the <img> tag with the href attribute set to the page url
LookupEntityImage	Returns an <img> tag but specifically for the specified entity. The ImageFileNameOverride or SKUparameter will allow this method to perform much faster than the aove LookupImage function if you use ImageFileNameOverride or SKU on your product's images. Either parameter can be and empty string if not used		<xsl:value-of select="aspdnsf:LookupEntityImage(ID, EntityName, DesiredSize, IncliudeATag)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ID	Integer	Category, Section, or Manufacturer ID
	EntityName	String	Literally: "Category," "Section", or "Manufacturer"
	DesiredSize	String	Literally: icon, medium, large
	IncliudeATag	Boolean (0/1)	Returns an <a> tag around the <img> tag with the href attribute set to the page url
ImageUrl	Returns the path (relative or full URL) to the specified entity image		<xsl:value-of select="aspdnsf:ImageUrl(ID, EntityOrObjectName, DesiredSize, FullUrl)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	ID	Integer	Product, Category, Section, or Manufacturer ID
	EntityOrObjectName	String	Literally: “Product”, “Category”, “Section”, or “Manufacturer”
	DesiredSize	String	Literally: icon, medium, large
	FullUrl	Boolean (0/1)	Determine whether the path is a relative path (e.g. images/product/icon/imgname.jpg) or a full URL (e.g. http://www.mydomain.com/images/product/medium/myimage.jpg)
ProductImageUrl	Returns the path (relative or full URL) to the specified product image. This method is faster than the above method if you are using ImageFileNameOverride or SKU as image name		<xsl:value-of select="aspdnsf:ProductImageUrl(ProductID, ImageFileNameOverride, SKU, DesiredSize, FullUrl)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	Product
	ImageFileNameOverride	String	Literally: “Product”, “Category”, “Section”, or “Manufacturer”
	SKU	String	Product SKU
	DesiredSize	String	Literally: icon, medium, large
	FullUrl	Boolean (0/1)	Determine whether the path is a relative path (e.g. images/product/icon/imgname.jpg) or a full URL (e.g. http://www.mydomain.com/images/product/medium/myimage.jpg)
ProductNavLinks	Returns the next-previous product navigation for products within the specified category or section		<xsl:value-of select="aspdnsf:ProductNavLinks()" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	CategoryID	Integer	Must only specified if the SectionID is not specified
	SectionID	Integer	Must only specified if the CategoryID is not specified
EmailProductToFriend	Returns an a hyperlink to a page to where you can email the product page		<xsl:value-of select="aspdnsf:EmailProductToFriend(ProductID, CategoryID)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	CategoryID	Integer	
ProductDescriptionFile	Returns the contents the product description file		<xsl:value-of select="aspdnsf:ProductDescriptionFile(ProductID, IncludeBRBefore)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	IncludeBRBefore	Boolean (0/1)	Return a tag before the contents of the file or not 1 = yes, 0 = no
ProductSpecs	Returns the contents a product specs file		<xsl:value-of select="aspdnsf:ProductSpecs(ProductID, IncludeBRBefore)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	IncludeBRBefore	Boolean (0/1)	Return a tag before the contents of the file or not 1 = yes, 0 = no
ProductRatings	Returns the product ratings section		<xsl:value-of select="aspdnsf:ProductRatings(ProductID, CategoryID, SectionID, ManufacturerID, IncludeBRBefore)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	
	CategoryID	Integer	
	SectionID	Integer	
	ManufacturerID	Integer	
	IncludeBRBefore	Boolean (0/1)	Return a tag before the HTML for the rating display, 1 = yes, 0 = no
FormatCurrency	Returns the currency value formatted for the specified locale		<xsl:value-of select="aspdnsf:FormatCurrency(CurrencyValue, LocaleSetting)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	CurrencyValue	Decimal	
	LocaleSetting	String	User or server default locale setting
GetSpecialsBoxExpandedRandom	Returns a formatted list of randomly selected items marked as being on special in the specified category		<xsl:value-of select="aspdnsf:GetSpecialsBoxExpandedRandom(CategoryID, ShowPics, IncludeFrame, Teaser)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	CategoryID	Integer	
	ShowPics	Boolean (0/1)	1 = Show product pictures, 0 = don't show pictures
	IncludeFrame	Boolean (0/1)	Create a frame around the list of specials, 1 = yes, 0 = no
	Teaser	String	Optional string rendered at the top of the list (inside the frame if it exists)
GetSpecialsBoxExpanded	Returns a formatted list of items marked as being on special in the specified category.		<xsl:value-of select="aspdnsf:GetSpecialsBoxExpanded(CategoryID, ShowNum , ShowPics, IncludeFrame, Teaser)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	CategoryID	Integer	
	ShowNum	Integer	The number of specials to show
	ShowPics	Boolean (0/1)	1 = Show product pictures, 0 = don't show pictures
	IncludeFrame	Boolean (0/1)	Create a frame around the list of specials, 1 = yes, 0 = no
	Teaser	String	Optional string rendered at the top of the list (inside the frame if it exists)
GetNewsBoxExpanded	Returns a list of news items in an option frame		<xsl:value-of select="aspdnsf:GetNewsBoxExpanded(ShowCopy, ShowNum, IncludeFrame, Teaser)" disable-output-escaping="yes"/>

	Argument name	**Data Type**	**Description**
	ShowCopy	Boolean (0/1)	Shows the entire content of the article
	ShowNum	Integer	The number of new items to show
	IncludeFrame	Boolean (0/1)	Create a frame around the list of specials, 1 = yes, 0 = no
	Teaser	String	Optional string rendered at the top of the list (inside the frame if it exists)
Decode	Decodes markup that might have been encoded during a conversion to XML		<xsl:value-of select="aspdnsf:Decode()" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	htmlContent	String	String to decode
ShowUpsellProducts	Returns a grid of products that have been deigated as upsell for the specified product		<xsl:value-of select="aspdnsf:ShowUpsellProducts(ProductID)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	The ProductID to show the upsell products for
ShowRelatedProducts	Returns a grid of products that have been deigated as upsell for the specified product		<xsl:value-of select="aspdnsf:ShowRelatedProducts(ProductID)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	ProductID	Integer	The ProductID to show the upsell products for
Decrypt	Returns the decrypted value of the data that was Encrypted using the ASPDNSF Encrypt method		<xsl:value-of select="aspdnsf:Decrypt(EncryptedData)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	EncryptedData	String	The data that you want decrypted
XmlPackage	Executes and returns the results of an XMLPackage. You must be careful doing this as it can cause an infinite loop		<xsl:value-of select="aspdnsf:XmlPackage(PackageName)" disable-output-escaping="yes"/>
	Argument name	**Data Type**	**Description**
	PackageName	String	The package name to run

Appendix B: XmlPackage Examples

Using all default storefront installation settings, we will add a custom display format to the "Test Section" department of the storefront. First, in the admin site, add these sub-sections under the "Test Section" dept:

- Bikes & Hikes
- Boots & Bikes
- Shovels & Trowels
- (whatever else you want, names don't matter)

Also, add descriptions to those sub-sections. Clear the cache on the storefront, to make sure those new subsections are now loaded. We will use the XmlPackage called "aspdnsf.xml.config" show below:

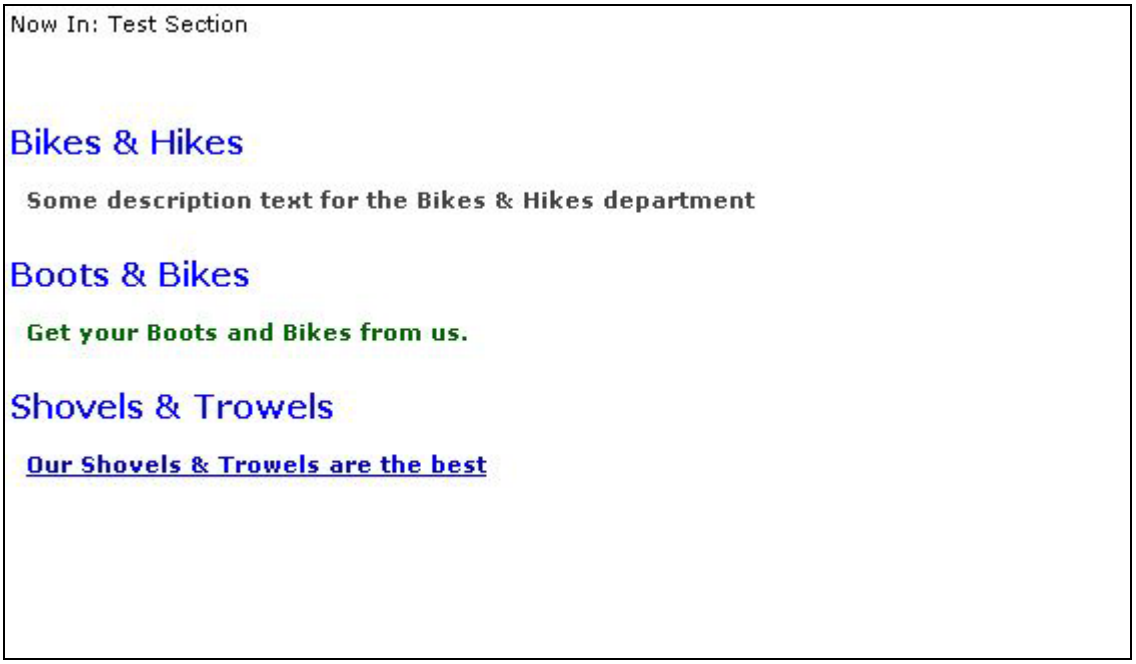
```
<?xml version="1.0" encoding="UTF-8" ?>
<package>
  <query name="Sections" rowElementName="Section">
    <sql>
      <![CDATA[
        select Name,Description, SEName from Section with (NOLOCK) where SectionID = @SectionID
      ]]>
    </sql>
    <queryparam paramname="@SectionID" paramtype="request" requestparamname="sectionid" sqlDataType="int"
                defvalue="0"  validationpattern="" />
  </query>
  <query name="SubSections" rowElementName="SubSection">
    <sql>
      <![CDATA[
        select Name, Description, DisplayOrder, SectionID, SEName from Section
        where ParentSectionID = @SectionID and Published >= @PublishedOnly and Deleted=0
        order by DisplayOrder,Name
      ]]>
    </sql>
    <queryparam paramname="@SectionID" paramtype="request" requestparamname="sectionid" sqlDataType="int"
                defvalue="0"  validationpattern="" />
    <queryparam paramname="@PublishedOnly" paramtype="runtime" requestparamname="PublishedOnly" sqlDataType="int"
                defvalue="0"  validationpattern="" />
  </query>
  <PackageTransform>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:aspdnsf="urn:aspdnsf">
      <xsl:output method="html" omit-xml-declaration="yes" />

      <xsl:template match="/">
        <xsl:apply-templates select="/root/Sections/Section" />
        <table border="0" width="100%" cellpadding="0" cellspacing="0">
          <xsl:apply-templates select="/root/SubSections/SubSection" />
        </table>
      </xsl:template>

      <xsl:template match="Section">
        <p align="justify"><xsl:value-of select="Description" disable-output-escaping="yes" /></p>
      </xsl:template>

      <xsl:template match="SubSection">
        <p align="justify"><b><a style="font-size:12pt;">
            <xsl:attribute name="href">
                <xsl:value-of select="aspdnsf:SectionLink(SectionID, SEName, 0)" />
            </xsl:attribute>
            <xsl:value-of select="Name" disable-output-escaping="yes" /></a></b><br /><br />
          &amp;#160;&amp;#160;<xsl:value-of select="aspdnsf:Decode(Description)" disable-output-escaping="yes" /><br />
        </p>
      </xsl:template>
    </xsl:stylesheet>
  </PackageTransform>
</package>
```

Save this XmlPackage in your skins/skin_1/XmlPackages directory.

Edit the “Test Section” dept in the admin site, and specify the Display Format of XmlPackage, and select the aspdnsf.xml.config package to be active in the XmlPackage select list. If you don’t see rob1.xml.config, you did NOT copy the file to the right place on your web server.

DEBUGGING NOTE: To enable debugging of XmlPackage output, set AppConfig:Xml.DumpTransform=true. This will cause all pages to output intermediate stages in the XmlPackage execution when an XmlPackage is being used, along with run-time parameters, SQL query parameterization, and output Xml/HTML from the package in a debug format. The debug files will ALSO be written to the /images directory or /admin/images directory. Do NOT forget to later TURN OFF this debug flag for production sites! It will cause errors in a multi-user environment, and slow the site down A LOT!

If you now browse to the Test Section output in your storefront, you will see a custom section page format produced, that will look “something” like this:



Remember, this is JUST the page contents are output. This is rendered WITHIN the skin.

What just happened?

Well, the storefront saw that the Test Section was set to use the rob1.xml.config package, so it loaded the rob1.xml.config file, added system parameters to it, executed the SQL queries defined in the package to create the working data sets, converted the data set to Xml, then Xslt (transformed) the Xml to HTML and wrote it to the screen

If you have set the debug flag you will see intermediate stage results as shown below:

XmlPackage that was loaded:

XmlPackages/aspdnsf.xml.config

```
<?xml version="1.0" encoding="UTF-8"?>
<package>
  <query name="Sections" rowElementName="Section">
    <sql>[CDATA[
       select Name,Description, SEName from Section with (NOLOCK) where SectionID
= @SectionID
     ]]</sql>
    <queryparam paramname="@SectionID" paramtype="request"
requestparamname="sectionid" sqlDataType="int" defvalue="0" validationpattern=""
/>
  </query>
  <query name="SubSections" rowElementName="SubSection">
    <sql>[CDATA[
       select Name, Description, DisplayOrder, SectionID, SEName
       from Section
       where ParentSectionID = @SectionID and Published >= @PublishedOnly and
Deleted=0
       order by DisplayOrder,Name
     ]]</sql>
    <queryparam paramname="@SectionID" paramtype="request"
requestparamname="sectionid" sqlDataType="int" defvalue="0" validationpattern=""
/>
    <queryparam paramname="@PublishedOnly" paramtype="runtime"
requestparamname="PublishedOnly" sqlDataType="int" defvalue="0"
validationpattern="" />
  </query>
  <PackageTransform>
    <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:aspdnsf="urn:aspdnsf">
      <xsl:output method="html" omit-xml-declaration="yes" />
      <xsl:template match="/">
        <xsl:apply-templates select="/root/Sections/Section" />
        <table border="0" width="100%" cellpadding="0" cellspacing="0">
          <xsl:apply-templates select="/root/SubSections/SubSection" />
        </table>
      </xsl:template>
      <xsl:template match="Section">
        <p align="justify">
          <xsl:value-of select="Description" disable-output-escaping="yes" />
        </p>
      </xsl:template>
```

The data document after all SQL Queries and Systems data have been combined:

aspdnsf.xml.config_store.runtime.xml

```
<root>
  <System>
    <IsAdminSite>False</IsAdminSite>
    <IsAdminSiteInt>0</IsAdminSiteInt>
    <PublishedOnly>1</PublishedOnly>
    <CustomerID>0</CustomerID>
    <CustomerLevelID>0</CustomerLevelID>
    <CustomerLevelName />
    <CustomerFirstName />
    <CustomerLastName />
    <CustomerFullName />
    <SubscriptionExpiresOn />
    <CustomerRoles />
    <IsAdminUser>false</IsAdminUser>
    <IsSuperUser>false</IsSuperUser>
    <LocaleSetting>en-US</LocaleSetting>
    <WebConfigLocaleSetting>en-US</WebConfigLocaleSetting>
    <SqlServerLocaleSetting>en-US</SqlServerLocaleSetting>
    <Date>11/30/2005</Date>
    <Time>1:28 AM</Time>
    <SkinID>1</SkinID>
    <AffiliateID>0</AffiliateID>
    <IPAddress>192.168.0.40</IPAddress>
    <QueryString>SectionID=1&amp;SEName=test-section</QueryString>
    <UseStaticLinks>false</UseStaticLinks>
    <PageName>showsection.aspx</PageName>
    <FullPageName>/version60/showsection.aspx</FullPageName>
    <XmlPackageName>aspdnsf.xml.config</XmlPackageName>
    <StoreUrl>http://dotnetstorefront6/version60/</StoreUrl>
  </System>
  <QueryString>
    <SectionID>1</SectionID>
    <SEName>test-section</SEName>
  </QueryString>
  <Form />
  <Session />
  <ServerVariables>
    <HTTP_HOST>dotnetstorefront6</HTTP_HOST>
    <HTTP_USER_AGENT>Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US;
rv:1.7.12) Gecko/20050915 Firefox/1.0.7</HTTP_USER_AGENT>
```

The XmlPackage Transform Document:

aspdnsf.xml.config_store.xfrm.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:aspdnsf="urn:aspdnsf">
  <xsl:output method="html" omit-xml-declaration="yes" />
  <xsl:template match="/">
    <xsl:apply-templates select="/root/Sections/Section" />
    <table border="0" width="100%" cellpadding="0" cellspacing="0">
      <xsl:apply-templates select="/root/SubSections/SubSection" />
    </table>
  </xsl:template>
  <xsl:template match="Section">
    <p align="justify">
      <xsl:value-of select="Description" disable-output-escaping="yes" />
    </p>
  </xsl:template>
  <xsl:template match="SubSection">
    <p align="justify">
      <b>
        <a style="font-size:12pt;">
          <xsl:attribute name="href">
            <xsl:value-of select="aspdnsf:SectionLink(SectionID, SEName, 0)" />
          </xsl:attribute>
          <xsl:value-of select="Name" disable-output-escaping="yes" />
        </a>
      </b>
      <br />
      <br />
          <xsl:value-of select="aspdnsf:Decode(Description)"
disable-output-escaping="yes" /><br /></p>
  </xsl:template>
</xsl:stylesheet>
```

The resulting output after transforming the data document using the above XSL Tranform document:

aspdnsf.xml.config transform result

```
<p align="justify" xmlns:aspdnsf="urn:aspdnsf"><br>
</p>
<table border="0" width="100%" cellpadding="0" cellspacing="0"
xmlns:aspdnsf="urn:aspdnsf">
  <p align="justify"><b><a style="font-size:12pt;" href="s-2-bikes-hikes.aspx">Bikes
& Hikes</a></b><br><br>
        <span style="font-weight: bold;">Some description text for the Bikes & Hikes
department</span><br>
<br></p>
  <p align="justify"><b><a style="font-size:12pt;" href="s-3-boots-bikes.aspx">Boots
& Bikes</a></b><br><br>
        <span style="color: rgb(0, 100, 0); font-weight: bold;">Get your Boots and
Bikes from us.</span><br>
<br></p>
  <p align="justify"><b><a style="font-size:12pt;"
href="s-4-shovels-trowels.aspx">Shovels & Trowels</a></b><br><br>
        <span style="text-decoration: underline; color: rgb(0, 0, 205); font-weight:
bold;">Our Shovels & Trowels are the best</span><br>
<br></p>
</table>
```

These steps allow you to examine exactly what happened in the XmlPackage execution.

Appendix C: Sample Packages

HELLOWORLD.XML.CONFIG:

This is a very simple package that produces output of "Hello World" for anon users, or "Hello TheirFirstNameGoesHere" output for a logged in user. Most of the param statements in this package are not needed, they are just shown for example. The only parameter that is actually used CustomerFirstName.

XmlPackages don't actually require a query. The system data is always there so there is always an xml data document to run the transform against.

```
<?xml version="1.0" standalone="yes" ?>
<package version="2.1" displayname="Hello World" debug="false" includeentityhelper="false">
  <PackageTransform>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                    xmlns:aspdnsf="urn:aspdnsf"
                    exclude-result-prefixes="aspdnsf">
    <xsl:output method="html" omit-xml-declaration="yes" />
    <xsl:template match="/">
      <xsl:choose>
        <xsl:when test="/root/System/CustomerFirstName!="">Hello <b><xsl:value-of
select="/root/System/CustomerFirstName"/></b></xsl:when>
        <xsl:otherwise><b>Hello World</b></xsl:otherwise>
      </xsl:choose>
    </xsl:template>
    </xsl:stylesheet>
  </PackageTransform>
  <SearchEngineSettings>
    <SETitle actionType="text">Hello World Page Title</SETitle>
    <SEKeywords actionType="text">Hello World SE Keywords</SEKeywords>
    <SEDescription actionType="text">Hello World SE Description</SEDescription>
    <SENoScript actionType="text">Hello World SENoScript</SENoScript>
    <SectionTitle actionType="text">Hello World Section Title</SectionTitle>
  </SearchEngineSettings>
</package>
```

ONELINEPRODUCTLIST.XML.CONFIG

This is a sample xml package that produces a list of products in a category or section, using a simple <li.> type of list, with links to each product page. Note how <ml> encoded fields are used, along with SQL statement run-time parameterization, etc…

```
<?xml version="1.0" encoding="UTF-8" ?>
<package version="2.1" displayname="Entity One Line Product List" debug="false" includeentityhelper="true">
  <query name="Products" rowElementName="Product">
    <sql>
      <![CDATA[
        exec aspdnsf_GetProducts
          @categoryID = @CatID,
          @sectionID = @SecID,
          @manufacturerID = @ManID,
          @localeName = @locale,
          @CustomerLevelID = @CustLevelID,
```

```
            @affiliateID = @AffID,
            @ProductTypeID = @ProdTypeID,
            @ViewType = 1,
            @pagenum = @pgnum,
            @pagesize = 16,
            @StatsFirst = 0,
            @publishedonly = 1,
            @ExcludePacks = 0,
            @ExcludeKits = 0,
            @ExcludeSysProds = 0,
            @InventoryFilter = @InvFilter,
            @sortEntityName = @entityname
        ]]>
    </sql>
    <queryparam paramname="@CatID" paramtype="runtime" requestparamname="CatID" sqlDataType="int" defvalue="0"
validationpattern="" />
    <queryparam paramname="@SecID" paramtype="runtime" requestparamname="SecID" sqlDataType="int" defvalue="0"
validationpattern="" />
    <queryparam paramname="@ManID" paramtype="runtime" requestparamname="ManID" sqlDataType="int" defvalue="0"
validationpattern="" />
    <queryparam paramname="@locale" paramtype="runtime" requestparamname="LocaleSetting" sqlDataType="varchar"
defvalue="en-US"  validationpattern="" />
    <queryparam paramname="@CustLevelID" paramtype="runtime" requestparamname="CustomerLevelID" sqlDataType="int"
defvalue="0"  validationpattern="" />
    <queryparam paramname="@AffID" paramtype="runtime" requestparamname="AffiliateID" sqlDataType="int" defvalue="0"
validationpattern="" />
    <queryparam paramname="@ProdTypeID" paramtype="runtime" requestparamname="ProductTypeFilterID" sqlDataType="int"
defvalue="1"  validationpattern="" />
    <queryparam paramname="@pgnum" paramtype="request" requestparamname="pagenum" sqlDataType="int" defvalue="1"
validationpattern="" />
    <queryparam paramname="@InvFilter" paramtype="appconfig"
requestparamname="HideProductsWithLessThanThisInventoryLevel" sqlDataType="int" defvalue="0"  validationpattern="" />
    <queryparam paramname="@entityname" paramtype="runtime" requestparamname="EntityName" sqlDataType="varchar"
defvalue=""  validationpattern="" />
  </query>

  <PackageTransform>

    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:aspdnsf="urn:aspdnsf" exclude-
result-prefixes="aspdnsf">
      <xsl:output method="html" omit-xml-declaration="yes" />

        <xsl:param name="LocaleSetting" select="/root/Runtime/LocaleSetting" />
        <xsl:param name="WebConfigLocaleSetting" select="/root/Runtime/WebConfigLocaleSetting" />
        <xsl:param name="ShowSubcatsInGrid"><xsl:value-of select="aspdnsf:AppConfig('ShowSubcatsInGrid')" /></xsl:param>
        <xsl:param name="SubcatGridCols"><xsl:value-of
select="/root/EntityHelpers/*[name()=/root/Runtime/EntityName]/descendant::Entity[EntityID=/root/Runtime/EntityID]/ColWidth"
/></xsl:param>
        <xsl:param name="EntityName" select="/root/Runtime/EntityName" />
        <xsl:param name="EntityID" select="/root/Runtime/EntityID" />
        <xsl:param name="WholesaleOnlySite"><xsl:value-of select="aspdnsf:AppConfigBool('WholesaleOnlySite')"/></xsl:param>

        <xsl:param name="BaseURL">
          <xsl:choose>
            <xsl:when test="aspdnsf:StrToLower(/root/Runtime/EntityName) = 'category'">c-<xsl:value-of
select="/root/Runtime/EntityID" />-<xsl:value-of select="/root/QueryString/sename" />.aspx</xsl:when>
            <xsl:when test="aspdnsf:StrToLower(/root/Runtime/EntityName) = 'section'">s-<xsl:value-of
select="/root/Runtime/EntityID" />-<xsl:value-of select="/root/QueryString/sename" />.aspx</xsl:when>
```

```
            <xsl:when test="aspdnsf:StrToLower(/root/Runtime/EntityName) = 'manufacturer'">m-<xsl:value-of
select="/root/Runtime/EntityID" />-<xsl:value-of select="/root/QueryString/sename" />.aspx</xsl:when>
            <xsl:when test="aspdnsf:StrToLower(/root/Runtime/EntityName) = 'library'">l-<xsl:value-of
select="/root/Runtime/EntityID" />-<xsl:value-of select="/root/QueryString/sename" />.aspx</xsl:when>
          </xsl:choose>
        </xsl:param>

        <xsl:param name="CurrentPage">
          <xsl:choose>
            <xsl:when test="/root/QueryString/pagenum"><xsl:value-of select="/root/QueryString/pagenum" /></xsl:when>
            <xsl:otherwise>1</xsl:otherwise>
          </xsl:choose>
        </xsl:param>



        <xsl:template match="/">
          <xsl:value-of select="aspdnsf:EntityPageHeaderDescription($EntityName, $EntityID)" disable-output-escaping="yes" />
          <xsl:value-of select="aspdnsf:EntityPageFilterOptions($EntityName, $EntityID, /root/Runtime/SecID, /root/Runtime/CatID,
/root/Runtime/ManID, /root/Runtime/ProductTypeFilterID)" disable-output-escaping="yes" />
          <xsl:call-template name="SubEntity" />
          <div style="text-align:right;"><xsl:value-of select="aspdnsf:PagingControl($BaseURL, $CurrentPage,
/root/Products2/Product/pages)" disable-output-escaping="yes" /></div>
          <xsl:apply-templates select="root/Products//Product" />
          <div style="text-align:right;"><xsl:value-of select="aspdnsf:PagingControl($BaseURL, $CurrentPage,
/root/Products2/Product/pages)" disable-output-escaping="yes" /></div>
        </xsl:template>


        <xsl:template match="Product">
          <xsl:param name="pName">
            <xsl:choose>
              <xsl:when test="count(Name/ml/locale[@name=$LocaleSetting])!=0">
                <xsl:value-of select="Name/ml/locale[@name=$LocaleSetting]"/>
              </xsl:when>
              <xsl:when test="count(Name/ml/locale[@name=$WebConfigLocaleSetting])!=0">
                <xsl:value-of select="Name/ml/locale[@name=$WebConfigLocaleSetting]"/>
              </xsl:when>
              <xsl:when test="count(Name/ml)=0">
                <xsl:value-of select="Name"/>
              </xsl:when>
            </xsl:choose>
          </xsl:param>

          <img align="absmiddle">
            <xsl:attribute name="src">skins/skin_<xsl:value-of select="aspdnsf:SkinID()"
/>/images/redarrow.gif</xsl:attribute>&#0032;
          </img>
          <a href="{aspdnsf:ProductandEntityLink(ProductID, SEName, $EntityID, $EntityName, 0)}">
             <xsl:value-of select="$pName"/>
          </a>
          <br />
        </xsl:template>

        <xsl:template name="SubEntity">
          <xsl:for-each
select="/root/EntityHelpers/*[name()=/root/Runtime/EntityName]/descendant::Entity[ParentEntityID=/root/Runtime/EntityID]">
            <xsl:variable name="scName">
              <xsl:choose>
```

```
                <xsl:when test="count(Name/ml/locale[@name=$LocaleSetting])!=0">
                  <xsl:value-of select="Name/ml/locale[@name=$LocaleSetting]"/>
                </xsl:when>
                <xsl:when test="count(Name/ml/locale[@name=$WebConfigLocaleSetting]) !=0 ">
                  <xsl:value-of select="Name/ml/locale[@name=$WebConfigLocaleSetting]"/>
                </xsl:when>
                <xsl:when test="count(Name/ml)=0">
                  <xsl:value-of select="Name"/>
                </xsl:when>
              </xsl:choose>
            </xsl:variable>

            <xsl:choose>
              <xsl:when test="$ShowSubcatsInGrid = 'true'">
                <table border="0" cellpadding="0" cellspacing="4" width="100%">
                  <xsl:if test="position() mod $SubcatGridCols = 1">
                    <tr>
                      <xsl:for-each select=". | following-sibling::*[position() &lt; $SubcatGridCols]">
                        <xsl:call-template name="SubCatCell">
                          <xsl:with-param name="scName" select="$scName" />
                        </xsl:call-template>
                      </xsl:for-each>
                    </tr>
                    <tr>
                      <td height="10" colspan="{$SubcatGridCols}">
                         
                      </td>
                    </tr>
                  </xsl:if>
                </table>
              </xsl:when>
              <xsl:otherwise>
                <p align="left">
                     <img border="0" src="{concat('skins/skin_', aspdnsf:SkinID(),
'/images/redarrow.gif')}">
                  </img>
                  <a href="{aspdnsf:EntityLink(EntityID, SEName, $EntityName, 0, '')}">
                    <xsl:value-of select="$scName" disable-output-escaping="yes"/>
                  </a>
                </p>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:for-each>
        </xsl:template>

        <xsl:template name="SubCatCell">
          <xsl:param name="scName"></xsl:param>

          <xsl:param name="URL">
            <xsl:value-of select="aspdnsf:EntityLink(EntityID, SEName, $EntityName, 0, '')" />
          </xsl:param>

          <td align="center">
            <a href="{$URL}">
              <xsl:value-of select="aspdnsf:LookupEntityImage(EntityID, $EntityName, 'icon', 0)" disable-output-escaping="yes" />
            </a>
            <br/>
            <a href="{$URL}">
              <xsl:value-of select="$scName" disable-output-escaping="yes"/>
```

```
            </a>
          </td>
        </xsl:template>
      </xsl:stylesheet>
   </PackageTransform>
</package>
```

HOMEPAGE.XML.CONFIG

This package approximates the default home page contents provided by the built-in storefront code. To use this package to control your home page contents area, instead of our built in code, you would copy this package to your /XmlPackages dir, and then set AppConfig:Xml.DefaultPage=homepage. If the store sees this, it will use the specified XmlPackage to COMPLETELY provide the home page contents area. You can therefore add other content you want, other database driven data, etc to your home page. This also shows how you could include the data from an RSS feed in your page (this uses a feed from ZDNet)

```
<package version="2.1" debug="false">
   <query name="FeaturedProductCount" rowElementName="row">
      <sql>
         <![CDATA[
            select count(*) as N from ProductCategory with (NOLOCK) where CategoryID=@FeaturedCategoryID and ProductID in
(select distinct ProductID from Product with (NOLOCK) where Deleted=0)
         ]]>
      </sql>
      <queryparam paramname="@FeaturedCategoryID" paramtype="appconfig" requestparamname="isFeaturedCategoryID"
sqlDataType="int" defvalue="0"  validationpattern="" />
   </query>

   <query name="NewsCount" rowElementName="row">
      <sql>
         <![CDATA[
            select count(*) as N from News with (NOLOCK) where Deleted=0
         ]]>
      </sql>
   </query>

   <webquery name="ZDNet" RetType="xml">
         <url>http://news.zdnet.com/2509-1_22-0-5.xml</url>
   </webquery>

   <PackageTransform>

      <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:aspdnsf="urn:aspdnsf">
         <xsl:output method="html" omit-xml-declaration="yes" />
         <xsl:param name="NumHomePageSpecials" select="aspdnsf:AppConfig('NumHomePageSpecials')"/>
         <xsl:param name="FeaturedCategoryID" select="aspdnsf:AppConfig('isFeaturedCategoryID')" />

         <xsl:template match="/">
            <xsl:value-of select="aspdnsf:Topic('HomeTopIntro')" disable-output-escaping="yes"/>
            <xsl:choose>
               <xsl:when test="$FeaturedCategoryID!=0 and /root/FeaturedProductCount/row/N!=0">
                  <xsl:choose>
                     <xsl:when test="$NumHomePageSpecials=1">
                        <br />
                        <xsl:value-of select="aspdnsf:GetSpecialsBoxExpandedRandom($FeaturedCategoryID, 1, 1, 'Checkout our
hottest specials!')" disable-output-escaping="yes"/>
```

```
                    </xsl:when>
                    <xsl:otherwise>
                      <br />
                      <xsl:value-of select="aspdnsf:GetSpecialsBoxExpanded($FeaturedCategoryID, $NumHomePageSpecials, 1, 1,
'Checkout our hottest specials!')" disable-output-escaping="yes"/>
                    </xsl:otherwise>
                  </xsl:choose>
                </xsl:when>
              </xsl:choose>
              <xsl:choose>
                <xsl:when test="/root/NewsCount/row/N!=0">
                  <br />
                  <xsl:value-of select="aspdnsf:GetNewsBoxExpanded(1, 3, 1, 'News and Announcements!')" disable-output-
escaping="yes"/>
                </xsl:when>
              </xsl:choose>

              <br /><br />
              <div>
                <span class="heading" style="font-weight:bold;">The latest From ZDNet</span><br /><br />
                <xsl:apply-templates select="/root/ZDNet/rss/channel/item"></xsl:apply-templates>
              </div>

            </xsl:template>

            <xsl:template match="item">
              <a href="{link}" target="_blank"><xsl:value-of select="title" /></a><br />
            </xsl:template>

          </xsl:stylesheet>
        </PackageTransform>
      </package>
```

PAGE XMLPACKAGE REFERENCES

The following site pages can use XmlPackages to replace their contents output:

Site/Page	AppConfig Name	Runtime Params Added To Package By Page	Notes
/default.aspx	AppConfig:XmlDefaultPage	None	None
/galleries.aspx	AppConfig:XmlGalleriesPage	None	None
/news.aspx	AppConfig:Xml.Page	None	None
/orderconfirmation.aspx	AppConfig:Xml.Page	OrderNumber	Order confirmation page still performs logic, such as sending receipts, etc. This package just replaces the output of the page if desired
/partners	AppConfig:Xml.PartnersPage	None	None
/receipt.aspx	AppConfig:Xml.ReceiptPage	None	None
/recentadditions.aspx	AppConfig:Xml.RecentAdditionsPage	None	None
/recentcomments.aspx	AppConfig:Xml.RecentCommentsPage	None	None
/requestcatalog.aspx	AppConfig:Xml.RequestCatalogPage	None	You are responsible for your own form handler probably, or

			to understand the one the store provides by default
/search.aspx	AppConfig:Xml.SearchPage	None	You are responsible for your own form handler probably, or to understand the one the store provides by default
/searchadv.aspx	AppConfig:Xml.SearchAdvPage	None	You are responsible for your own form handler probably, or to understand the one the store provides by default
/shoppingcart.aspx	AppConfig:Xml.`ShoppingCartPageHeader` and ApPConfig:`Xml.ShoppingCartPageFooter`	None	These two XmlPackages could be used to add data driven header/footers to the cart page, but the cart still displays itself
/showcategory.aspx	N/A (Defined for category in admin site)	None	None
/showsection.aspx	N/A (Defined for section in admin site)	None	None
/showlibrary.aspx	N/A (Defined for library in admin site)	None	None
/showmanufacturer.aspx	N/A (Defined for manufacturer in admin site)	None	None
/showproduct.aspx	N/A (Defined for product in admin site)	None	None
/showdocument.aspx	N/A (Defined for document in admin site)	None	None
/sitemap.aspx	AppConfig:Xml.SiteMapPage	None	None
/staff.aspx	AppConfig:Xml.StaffPage	None	None
/tableorder.aspx	AppConfig:Xml.TableOrderPage	None	You are responsible for your own form handler probably, or to understand the one the store provides by default
/wishlist.aspx	AppConfig:Xml.WishListPage	None	You are responsible for your own form handler probably, or to understand the one the store provides by default
/admin/splash.aspx	AppConfig:Xml.SplashPage	None	None