

---

*<Tagger Cat />*



*Visual Model Driven JSP Application  
Development*

# Template Developer's Guide

Version 1.0

### Document Summary

<b>Title:</b>	Developer's Guide	<b>Document No.</b>	SLD - 194
<b>Department:</b>	Information Technology		
<b>Prepared by:</b>	Grant Genereux		
<b>Approval 1:</b>			
<b>Approval 2:</b>			
<b>Approval 3:</b>			

### Revision Record

Rev	Date	Page	Subject	Approval
1	01/10/06	All	Initial Revision	

Copyright © 2006-2007 taggercat.com

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms: You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how taggercat.com disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of taggercat.com. taggercat.com reserves any and all rights to this documentation not expressly granted above.

<b>Document No.</b> SLD – 195		Title: Templates Developer's Guide
<b>Rev:</b> 1	<i>Page ii</i>	<i>Created on: Aug. 10, 2007</i>

## Table of Contents

<b>1. Introduction .....</b>	<b>2</b>
1.1 What are Tagger Cat Templates? .....	2
<b>2. The Templates Application and the Dreamweaver Tagger.....</b>	<b>4</b>
2.1 Template Composition.....	7
2.2 The Dreamweaver Tagger's use of Templates.....	7
2.3 Template Regions and the tc:template tag.....	9
2.3.1 Design Time Template injection .....	9
2.4 Design Time Template Rebuilding .....	10
2.4.1 Template Rebuilding when the Model or Template definition changes .....	10
2.4.2 Revising Existing Template Regions .....	11
<b>3. The Templates Tag Library.....</b>	<b>12</b>
<b>4. Dynamic Template Regions.....</b>	<b>14</b>
4.1 Differences between Dynamic and Static Template regions .....	14
4.1.1 Statically Injected Template Regions.....	15
4.1.2 Dynamic Template Regions.....	15
4.1.3 When Dynamic Templates Fail.....	16

# 1. Introduction

Reusability is the key to improving productivity. In larger database applications we usually have repeated common functionality, such as searching, views, filtering, editing, etc.. You encounter the same design problems or scenarios where only the names of the entities and the properties change. When you present this functionality in your application, you don't want to design what is basically the same page over and over again. Doing so is not productive, and is simply not maintainable in larger applications.

Tagger Cat provides a template infrastructure and sample templates to build the re-occurring functionality from the application's model definition. The benefits of model driven templates are apparent both in the savings in design time, and longer term as you evolve your application. You can consider templates to be similar to reusable code libraries used in application development.

Even though we encourage you to use Tagger Cat's templates, they are not at all required. They are just a productivity gain. You can still easily build Tagger Cat applications without taking advantage of the templates. As a matter of fact, when building one-of-a-kind pages, we often initially build a page using the templates, and then detach the template regions, and then maintain the resulting page by hand.

## 1.1 What are Tagger Cat Templates?

Tagger Cat templates generate the Model Driven regions of your application. Rather than use Velocity, Freemarker, Jet etc, as template frameworks, Tagger Cat uses JSP as a template solution. Some people are initially thrown off by using JSP to generate JSP; since they expect JSP to be used exclusively for generating HTML content. However, JSP is a superb template language for generating all kinds of content. For Tagger Cat, we mostly use JSP templates to generate JSP code at design time. However, we also use JSP templates to generate XML, ATOM, etc..

Obviously, since you are using JSP to develop your application, you can apply the same skill set to develop and maintain JSP based templates.

Here are some benefits of using JSP as a template solution.

- ✓ There is very little new to learn, since it is JSP and a small tag library.
- ✓ JSP templates are built and tested as regular JSP pages.
- ✓ You can debug your templates just as you do regular JSPs.
- ✓ You can use any JSP tag library, including of course, JSTL with your templates.
- ✓ You can use JSP 2.0 EL expressions
- ✓ You host your templates in a separate web application, so there is no developer desktop to synchronize.
- ✓ For more advanced solutions; your templates can themselves be complete Tagger Cat web applications.
- ✓ You can call the same templates from desktop applications.
- ✓ Template support multi-value parameters.
- ✓ You can use the Template tag library to read your applications metadata at runtime.

## ***Tagger Cat Templates***

There is a set of standard templates provided with Tagger Cat. The standard templates cover the more common use cases; such as:

- ◆ Searching
- ◆ Table Views
- ◆ Scalar Views
- ◆ Editing
- ◆ Navigations

However, the standard templates are really just samples. There is no question that you will need to modify the standard templates to meet your specific needs. In addition to the standard ones, you also need to develop your own templates. Therefore, to get the full advantage of the templates, you need to make the investment in building them and maintaining them.

However, this investment will definitely payoff in a many fold gain in productivity. In addition to the initial productivity gain of using model driven development during the application development, there is a secondary benefit. Template development significantly reduces the impact of model changes on the presentation tier. Therefore, your development teams can respond to changes in requirements much faster than if each page is custom developed.

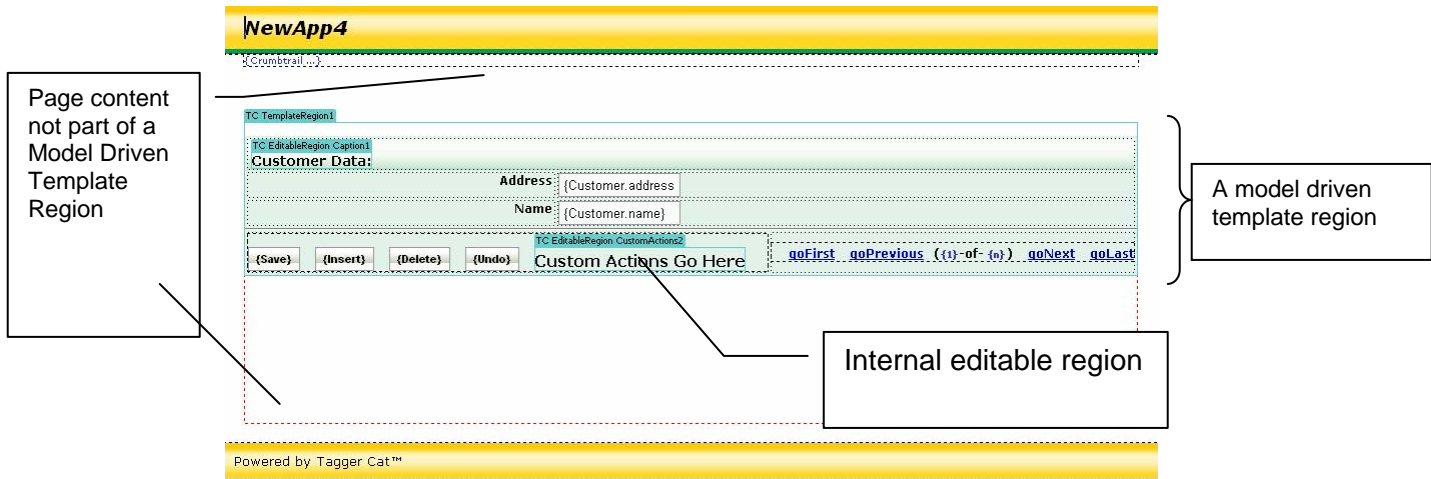
The templates are most often used to generate page content at design time. But, one of the most important aspects of using JSP based templates is that they can also be used at runtime. Your templates can be used at runtime to generate very dynamic page content. Used at runtime, the templates are most commonly applied in conjunction with Ajax calls and for end-user defined pages.

O.K. that's the sales pitch for using JSP based templates for generating JSP. Next we will discuss some of the specific benefits of using Tagger Cat's templates within Dreamweaver.

Firstly, it is important to understand our term *Model Driven Regions*. What we mean by this is that typically, only the regions of your application pages that reflect the Model are maintained in template definitions. Although, you can generate complete JSP pages using the Templates, you will still develop regions within those pages that are driven by the application's model. This is critically important because it frees page designers to use any markup and content outside of the template regions that they want.

The following figure shows a model driven region within Dreamweaver's design view. The template region is shown with a tabbed border at design time (not at runtime). Content outside of that region can be designed and maintained independent of the template regions.

Figure 1 Model Driven Region at Design Time



1. Template regions are driven from your model's metadata and the template definition.
2. Page designer instantly see the generated content within the page at design time.
3. Template regions can be rebuilt, and the non-template page sections are completely maintained.
4. When your template definitions change, you can rebuild all pages to reflect that change. Similarly, when your model's definition changes you can easily rebuild all your pages to reflect that.
5. Template regions can be set to dynamic, to automatically reflect changes in the model's definition (without rebuilding pages).
6. Template regions can have internal editable content regions. The editable content regions are maintained when their containing template region is rebuilt.
7. Specific form element types can be generated based upon the metadata, or explicitly selected at design time.
8. Template development is in the domain of page authors, and typically requires very little (~zero) Java code.

The templates read your application model in the same way that it is read at runtime.

## 2. The Templates Application and the Dreamweaver Tagger

The default configuration is to have a separate web application host your design time templates. The templates application is usually named **YourAppTemplates**. You will typically have one template application corresponding to your application under development. This one-to-one configuration of a template application per development application is recommended, but it is not required. You can deploy your design time templates to the same context as your development application if wanted.

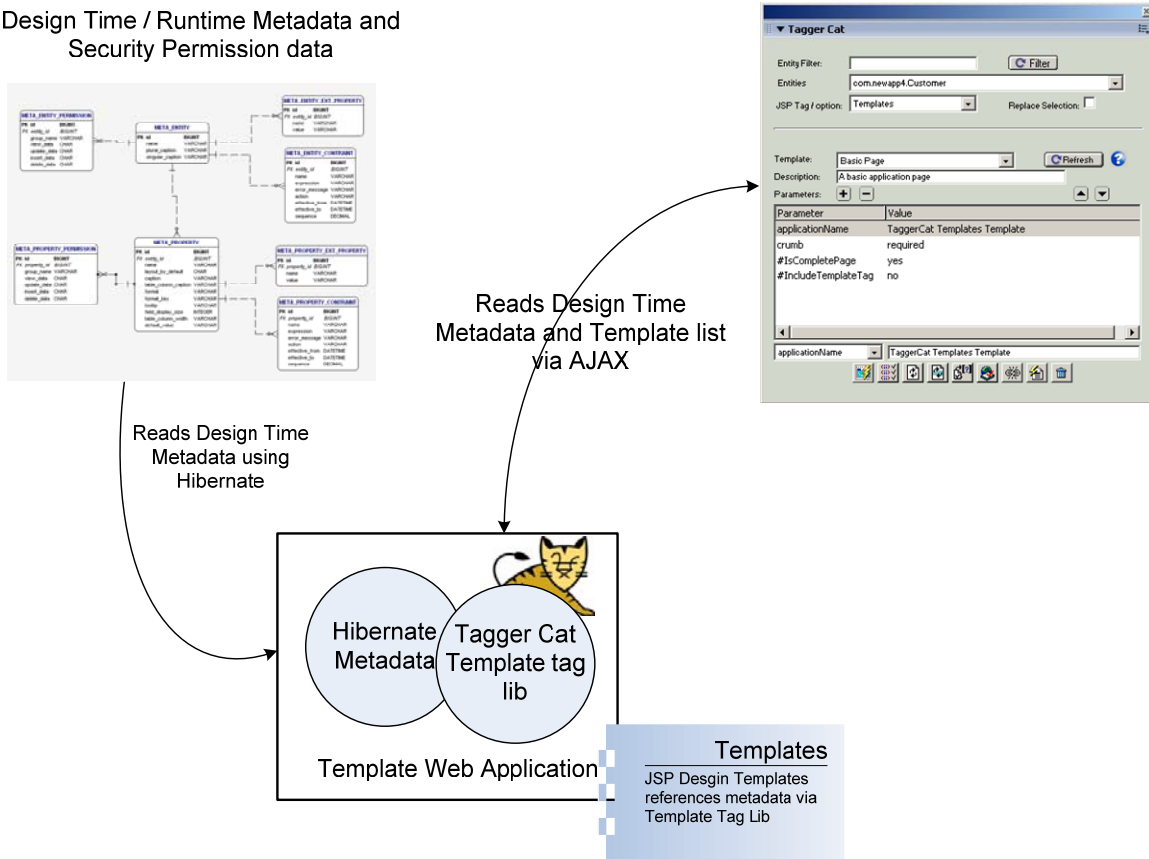
Since the Templates are a separate web application, we recommend that you create a separate Dreamweaver project for it.

## Tagger Cat Templates

In practice we find that the templates are modified quite a bit as the application evolves. Having a separate Dreamweaver project for editing the templates is very convenient. We prefer editing our JSPs with Dreamweaver over Eclipse; but that is just a personal preference.

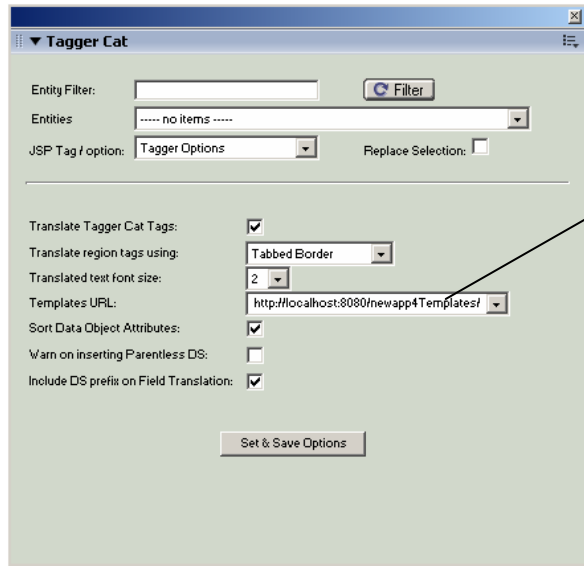
**Figure 2 The Dreamweaver Tagger calls the Templates application for all metadata**

Design Time / Runtime Metadata and Security Permission data



## Tagger Cat Templates

Figure 3 The Dreamweaver Tagger is configured with the URL of the templates application.

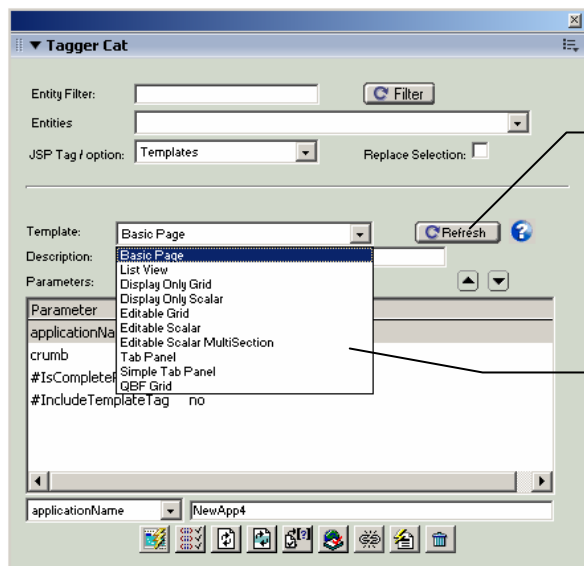


Set the Template URL in the options panel.

The available templates are listed in the templitelist.xml file. The templitelist.xml file is also configured as the welcome file for the templates application. Setting your browser's address to the URL <http://localhost:8080/MyAppTemplates> will serve the template list file.

The templates web application is not just used to read and process template requests. It is also used to read the applications metadata, and for all the Tagger's code generation panels.

The available templates are listed in the Dreamweaver Tagger's template panel.



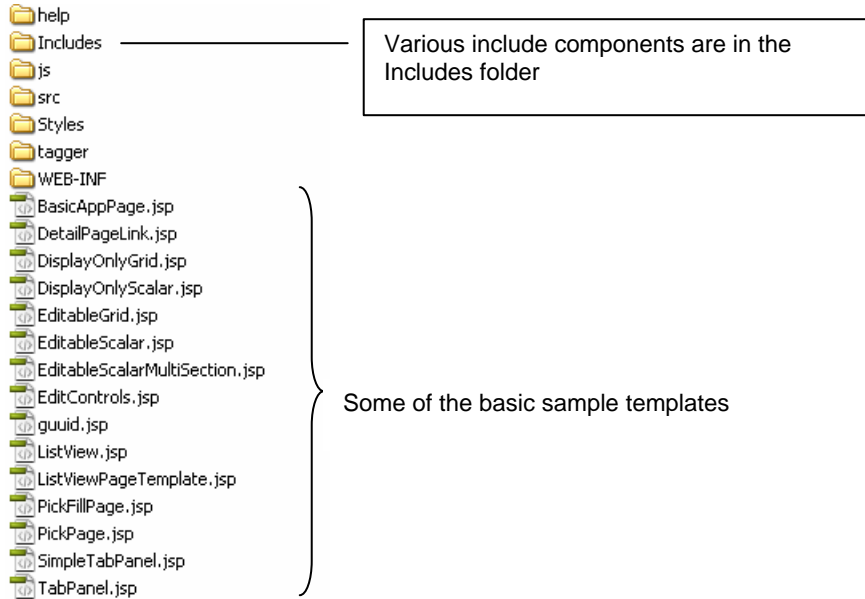
The refresh button re-reads the templates definitions.

The list of available templates

## 2.1 Template Composition

Just as you would develop regular JSP pages using reusable components, the sample templates are also built using a combination of includes and conditional content. The common included components are included in the /Includes folder of the template application.

Figure 4 A typical templates application folder



On a macro scale, the templates usually create a table, or other significant page section. On a micro scale, the templates can generate individual form elements for each model property. Since the data fields and form elements are reusable components, the macro scale templates delegate the generation of individual field form elements to a page called /EditControls.jsp.

Changes to the /EditControls.jsp page will have a global effect on all the higher level templates that generate form elements.

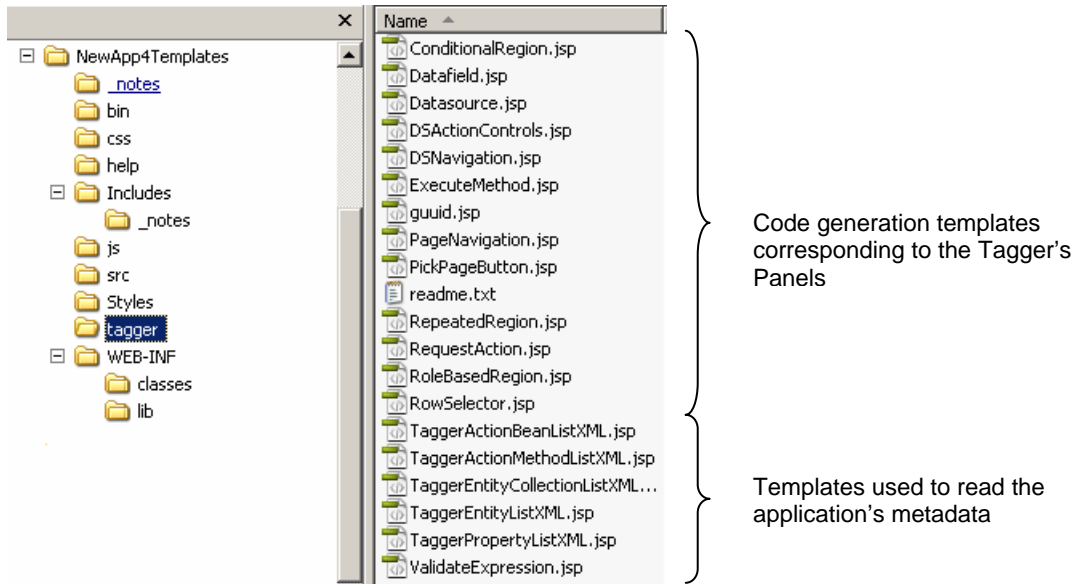
## 2.2 The Dreamweaver Tagger's use of Templates

When you are using the Dreamweaver Tagger to inject templates into your pages at design time, the Tagger makes an Ajax style call to the templates web application. The generated content is usually injected into the page at the current cursor location or replaces the existing selected template region.

However, the other panels of the Dreamweaver Tagger also make use of the templates application. All of the code generation of the Dreamweaver Tagger is generated in the Templates application, and is called via Ajax style calls to the templates application.

There is a separate folder call /tagger in the templates web application. In the /tagger folder there is a JSP page corresponding to each of the code generation panels of the Dreamweaver Tagger. Therefore, just as with the main templates, you have complete control over the code generation of Dreamweaver Tagger.

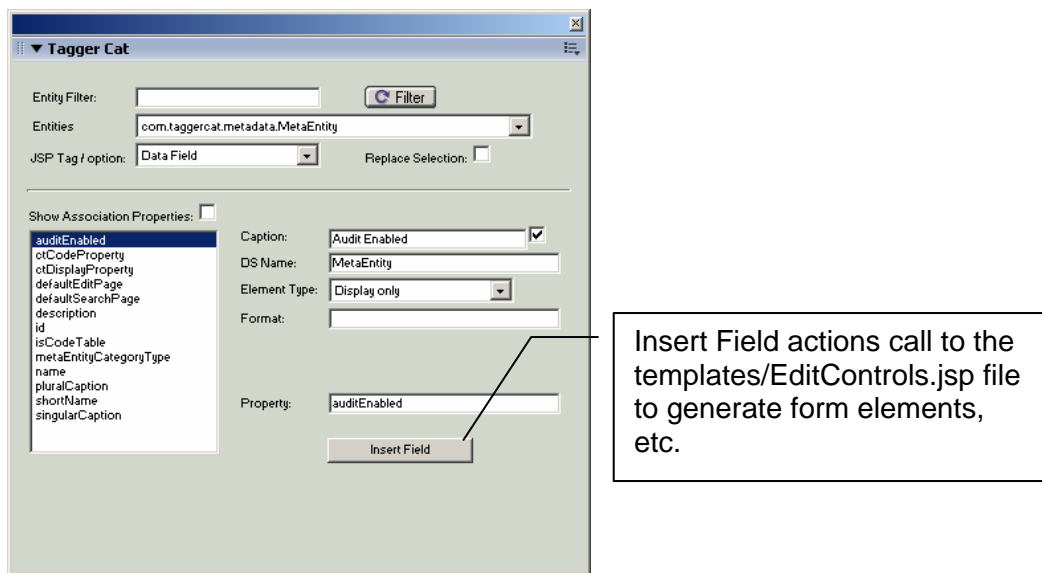
Figure 5 Tagger Code Generation Templates



As shown in the previous figure, the Template's web application /tagger folder contains a set of files name tagger..xml.jsp. These files are used to generate the Ajax response to the Tagger when reading the Entities, and properties from the Metadata. You would not typically need to modify these files, but they are good example of reading the metadata, and generating XML using JSP.

The Data Field panel of the Dreamweaver Tagger is used to inject "one field at a time" into your JSP page. The Data Field panel also calls the templates/EditControls.jsp file to generate the field form elements.

Figure 6 The Data Field Panel



## 2.3 Template Regions and the tc:template tag

When a template region is generated into a page, it is usually surrounded by a tc:template tag.

For example:

```
<tc:template guuid="b5296a65_91a7_336d_b64a_a77466b52faf"
uri="/EditableScalar.jsp?entityName=com.newapp4.Customer&datasource=Customer&selectedFields=required&selectedFieldsFieldTypes=&breakAfter=required&qbFields=optional&qbBreakAfterFields=optional" >
  <table class="detailDataTable" >
    <tr>
      <th colspan="24" class='dataTableHeader' ><tc:editable name="Caption">Customer
Data:</tc:editable></th>
    </tr>
    <tr>
      <th class="dataTableFieldCaption">Address</th>
      <td ><tc:textfield dsn="Customer" field="address"
size="20" maxlength="50" format="" title="" /> </td>
    </tr>
  ...
</tc:template>
```

The <tc:template ... > tag serves multiple functions:

1. It uniquely identifies the template region; using a system generated uuid
2. Its URI attribute includes the name of the template resource, and the definition of the parameters used to generate its content at design time
3. It is used when the template is processed at runtime
4. It is used by the Tagger to allow it to be rebuilt, and translated at design time

The default runtime functionality of the <tc:template > tag is to process its body content as JSP. Therefore, there is no reference to the tc:template tag that end users can see if they view the page source.

### 2.3.1 Design Time Template injection

At design time a template is injected into a page using the Tagger's template panel. This is quite a straight forward process; it builds an HTTP URI and a query string.

1. the URI is the name of the template resource ( .jsp ) being called
2. the query string is the entityName, and all named parameters and their values

When submitted, the host name, and application context of the templates application are pre-pended to the URI, and the complete URL is processed as a web request.

The content of the response is injected into the page at either the cursor location if it is outside of an existing template region; or replaces an existing template region if one is selected.

The uri attribute of the template region is set ( or updated ) to reflect the current parameters.

## **2.4 Design Time Template Rebuilding**

There are several scenarios when you will want to rebuild one or more template regions. There are some subtle, but important differences that you need to be familiar with here to know when and how to update your applications JSP pages.

- 1) Your template definition has changed. You have added or revised templated content, functionality etc. We find that this is the most common scenario.
- 2) Your model has changed, such as field captions, data types, field sizes, etc. But, the actual selection of fields within the template region have not changed.
- 3) You want to add or remove a field from the template region; or change its order, or form element type.

The next section covers rebuilding template regions in scenarios 1 and 2.

### **2.4.1 Template Rebuilding when the Model or Template definition changes**

Rebuilding template regions at design time is quite easy. You have several options:

#### ***In the Dreamweaver Design View***

To rebuild an individual template region, place the cursor within the template region, and select Rebuild Template from the Tagger Cat context menu.

To rebuild all template regions, select Rebuild Page Templates from the Tagger Cat context menu.

#### ***From the Dreamweaver Site Panel***

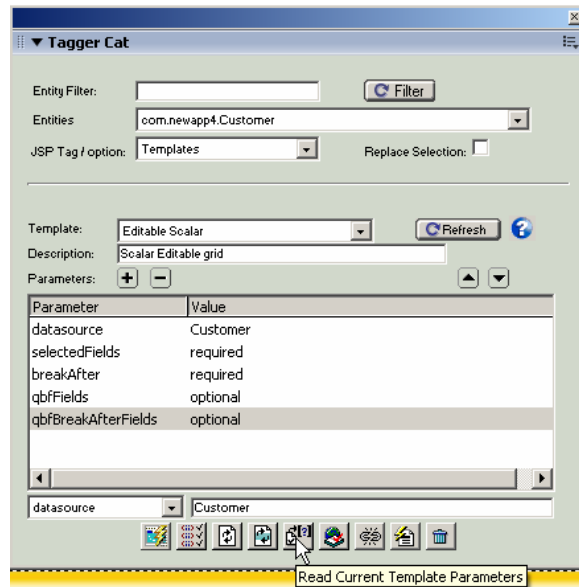
Select the files to rebuild the template regions in; then select Rebuild Tagger Cat Templates from the context menu.

## Tagger Cat Templates

### 2.4.2 Revising Existing Template Regions

You can easily revise (changing field selections etc.) an exiting template region. In design view:

1. Place the cursor within the existing template region.
2. Select the Read Current Template Parameters button from the Tagger's template panel.



3. Revise the field selection as needed
4. Re-inject the template definition into the page at the same location as the existing one.

### 3. The Templates Tag Library

The templates application uses the Template Tag library to read the metadata, and process the selected fields. Here is a summary of the most commonly used tags.

**Table 1 Template Tag Library Summary**

Tag Summary	
<a href="#">removeLines</a>	Removes blank lines from evaluated body. This is a cosmetic formatting tag.
<a href="#">datafield</a>	Creates a input field for a datasource attribute.
<a href="#">forEachFieldList</a>	This tag is used to iterate over fields lists.
<a href="#">forEachField</a>	This tag is used to iterate over a list of fields.
<a href="#">forEachTableRow</a>	This tag is used to create a nicely wrapped HTML table from a list of fields.
<a href="#">forEachTableRowField</a>	This tag is used to iterate over a list of fields that will make up a single row of an input table.
<a href="#">datafieldCaption</a>	This tag returns the Caption for the current datafield. If the field has no specified caption, then the field's name is used as the caption. If the field name is used for the caption, then by default, the name is pretty printed.
<a href="#">setEntityVars</a>	Sets request scope vars for current entity name, and object.
<a href="#">setAllAttributesVar</a>	This tag builds a comma separated list of all field names in the current entity and sets it as a request scope attribute.
<a href="#">oneLine</a>	Unwraps multiple lines into one line. This is a cosmetic formatting tag.
<a href="#">verbatim</a>	Does not interpret body content, and scriptlet code; it is included verbatim.

The best way to explain the templates tag library is by example.

**Listing 1 Display Only Scalar Template**

```

1.  <!-- This template creates a display only Scalar form. -->
2.  <!-- @version $Revision: #2 $ $DateTime: 2006/12/21 08:56:36 $ -->
3.  <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
4.  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
5.  <%@ taglib prefix="t" uri="http://www.taggercat.com/TemplateTags" %>

6.  <t:removeLines> <!-- this tag will remove any resulting blank lines -->

7.  <t:setEntityVars />
8.  <t:setAllAttributesVar />
9.  <table class="detailDataTable" >
10. <tr>
11. <th colspan="24" class='dataTableColHeader' >${requestScope.metaEntity.singularCaption} Data:</th>
12. </tr>
13. <!-- Create the QBF Header if any -->
14. <jsp:include page="/Includes/QBFHeaderSection.jsp" />
15. <t:forEachTableRow selectedFields="${allSelectedOrDefaultFields}"
breakFields='${param.breakAfter}' >
16. <tr>
17. <t:forEachTableRowField varStatus="status" >
18. <th width="${averagePercentColWidth}%" class="dataTableFieldCaption">
19. <t:datafieldCaption />&nbsp;  </th>
20. <td width="${averagePercentColWidth}" ${COLSPAN} >&nbsp;  <td>
21. <t:datafield editable='false' tooltips='true' maxSize='20' /></td>
22. </t:forEachTableRowField>
23. </tr>

```

## Tagger Cat Templates

```
24. </t:forEachTableRow>
25. </table>
26. <jsp:include page="/Includes/DisplayOnlyNavBar.jsp" />
27. </t:removeLines>
```

The following is a discussion of the above sample template.

The preamble from lines 1 – 5 should be self explanatory.

```
6. <t:removeLines> <%-- this tag will remove any resulting blank lines --%>
```

This is a formatting tag that will remove resulting white space from the generated template. This is just for improving the readability of the generated code, and has no other function.

```
7. <t:setEntityVars />
```

This sets the page and request scope variable named **metaEntity** for the selected entity. This tag should be included in all templates that process a field list, or reference meta entity properties.

```
8. <t:setAllAttributesVar />
```

This sets the page and request scope variable named **allSelectedOrDefaultFields** with a list of all the properties from the selected entity that are marked as *Layout by Default*.

```
9. <table class="detailDataTable" >
10. <tr>
11. <th colspan="24" class='dataTableColHeader' >${requestScope.metaEntity.singularCaption}
    Data:</th>
12. </tr>
13. <%-- Create the QBF Header if any --%>
```

This is standard HTML markup, with an EL expression to read the Singular caption from the current metaEntity.

```
14. <jsp:include page="/Includes/QBFHeaderSection.jsp" />
```

This includes a QBF selection ( if QBF fields were selected ).

```
15. <t:forEachTableRow selectedFields="${allSelectedOrDefaultFields}"
    breakFields='${param.breakAfter}' >
```

The forEachTableRow is where things start to get interesting. This is an iteration tag; but it is doing more than mundane iteration. The tag takes the selected field list and determines how many table rows are needed to create the required table. The tag uses the fields listed in the *param.breakAfter* expression to decide how to apply the column breaks to create the table rows. It also calculates the average % column widths for the table.

Also note that the selectedFields attribute is set with the expression: *allSelectedOrDefaultFields*.

What this is doing is if the template request did not include an explicit field list, then use the one created with all the default (layout by default) properties. Otherwise, use the explicit list.

```
17. <t:forEachTableRowField varStatus="status" >
```

The forEachTableRowField tag is used to iterate over the field list that make up each row in the table. Then in each iteration we include the field (property's caption) and the field itself.

## Tagger Cat Templates

```
19. <t:datafieldCaption />&nbsp;</th>
20. <td width="{averagePercentColWidth}" {COLSPAN} >&nbsp;<
21. <t:datafield editable = 'false' tooltips = 'true' maxSize = '20' /></td>
```

The `<t:datafield ..>` tag is the real workhorse of the template tag library since it is used in nearly every template. It is used to indirectly generate all kinds of form elements, as well as read-only values. The exact type of form element generated for each property depends upon the template request, and the metadata. However, as stated before, the `<t:datafield ..>` does not directly generate the content for each field's tag. It only sets up some scope variables, and then delegates the call to the `/EditControls.jsp` page.

Therefore, as with every other aspect of the templates, you can control exactly what is being generated by customizing the `/EditControls.jsp` page.

## 4. Dynamic Template Regions

The most common scenario is to apply templates at page design time. When a template region is added to a page, the generated template content is injected into the page at the current cursor location. However, you can have the templates regions function dynamically at runtime. When used dynamically the template can respond to changes in the model's definition without needing to rebuild pages within Dreamweaver. The templates can also respond to programmatically modified URIs.

### 4.1 Differences between Dynamic and Static Template regions

Typically, the template content is bounded by an owning Tagger Cat template tag.

```
<tk:template guuid="7dfgsdfg37373" uri="/ListView.jsp?entityName=com.cts.model.Deployedserver&datasource=Dep
<input type="hidden" name="DeployedServer.rowID" id="DeployedServer.rowID" value="" />
<div cl...
<div cl...
</tk:template>
```

The template tag has the following attributes:

#### *guuid*

The `guuid` attribute is generated by the Tagger, and is used to uniquely identify this template instance. The `guuid` attribute value is retained when the template region is rebuilt ( or re-injected ) using the Tagger.

#### *dynamic*

The `dynamic` attribute can be used to specify that the templates body content be ignored at runtime, and replaced by the resulting content of reprocessing the uri as a web request. See the discussion below on the details on how this attribute affects the tags processing.

#### *uri*

The `uri` includes the name of the top level jsp template definition file, and additional request parameters passed to the template.

### 4.1.1 Statically Injected Template Regions

The default functionality of the templates is to create HTML and JSP content that is driven by Tagger Cat's metadata. The statically generated content usually includes JSP HTML markup, Tagger Cat tags, as well as JSP 2.0 expressions.

It is important to remember that the Template is generating JSP tags that are to be later processed as such at runtime.

For example, say we want the following JSP tag:

```
<tc:dataFieldValue dsn="DeployedServer" field="region" format="" />
```

generated into the page, and not the resulting value of say "North America".

Therefore, the templates do not include Tagger Cat's taglib directives.

```
<%@ taglib prefix="tc" uri="http://www.taggercat.com/taglibs/taggercat" %>  
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>
```

By not including the above directives, the JSP template treats the Tagger Cat tags as non-jsp markup and includes it in the resulting template region.

### 4.1.2 Dynamic Template Regions

It is possible to also use the templates dynamically, and not statically as described above. But, be aware that this is an advanced topic.

In Tagger Cat's standard tag library the *dynamic='true'* attribute can be set on the template tag to make it dynamic.

Without the *dynamic='true'* attribute, the standard functionality of the template tag is to include its body content as jsp. Therefore, this effectively removes the `<tc:template ... >` tag from the resulting generated HTML markup sent to the user's browser. In other words, end users don't see a big long uri attribute, and the internals of how the page was generated.

When the *dynamic='true'* attribute is set, the template tags ignores its statically injected body content, and instead includes the resulting output of re-issuing the tag's uri attribute to the server. Therefore, when the *dynamic='true'* attribute is set, Tagger Cat's template tags acts very similar to a standard `<jsp:include ...>` tag, and very much like the `<c:import ...>` tag of the JSTL.

However, the template tag does not re-issue the tags uri to the same server that was used for static injection. Rather, the tag needs to send the request to the current application context. Therefore, for the *dynamic='true'* attribute to work, you need to have a runtime version of the template file in your application context.

Why two versions? Well when we are statically injecting the content we want markup such as:

```
<tc:dataFieldValue dsn="DeployedServer" field="region" format="" />
```

## Tagger Cat Templates

and expressions such as `${Customer.name}` to end up in our target jsp page.

On the other hand, when we are in a dynamic mode, we want the *result* of these tags and expressions in the output stream being sent to the user's browser. For example, the result might be 'North America' and 'Royal Builders'.

Now, additionally, we typically develop with the templates application being an external, separate application context from the application we are building. Therefore, to run the templates dynamically, we need the dynamic template to be running in the same context as our application. That way it knows how to get the value: 'North America' for the included tag.

Fortunately, the differences between the runtime ( dynamic ) and the static versions of the template files are quite minor. It is mostly just including the two tag lib declarations in the dynamic version of the page not escaping JSP expressions.

```
<%@ taglib prefix="t" uri="http://www.taggercat.com/TemplateTags" %>
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>
```

So, the complexity of supporting the *dynamic='true'* is that you need to:

- a) Maintain a separate template definition file in the runtime application corresponding to static ( injected ) version.
- b) Dealing with things when something breaks ( debugging ).

So, why and when would you want to use dynamic templates? Well firstly remember that the *dynamic='true'* setting means that you are using just regular old JSP, it is just slightly more elaborate because it is probably reading metadata to do its job. As stated above, the template tag with *dynamic='true'* is nearly identical to using a `<c:import ...>` tag.

But, the typical application of *dynamic='true'* is to support AJAX functionality, user defined page regions, and the progressive search.

What the *dynamic='true'* does not do is update the static body content in your application's source .jsp files. In other words, it does not re-inject the design time template regions back into the source page.

### 4.1.3 When Dynamic Templates Fail

Similarly to using a `<c:import ...>` tag, if a dynamic template fails for any reason Tomcat is not great at reporting the error (this condition might be by design, since the application is just making a call to an external resource, and getting an empty response back might be valid). So, when a dynamic template fails you might end up with a page with an empty content region in it, and no error message being displayed in the browser.

However, since you are running your template ( the external resource ) in the same context as your application, you will usually see some error message in the Tomcat console, and log files. Additionally, if the error originates in the Tagger Cat layer you will usually see error messages listed in Tagger Cat's log file(s).

## ***Tagger Cat Templates***

With or without meaningful error messages you can apply the same debugging techniques to dynamic templates as you do to regular JSPs. Remember, when templates are dynamic they are *exactly the same* as regular JSPs includes.