

Advanced Model-Glue

Doug Hughes, President - Alagad Inc.

Overview

- I'm assuming you already know Model-Glue
- This session will introduce you to a range of advanced features in Model-Glue
- Model-Glue has a lot of hidden features
- We'll cover a *subset* of some of the features

We'll Cover

- The Include Tag
- Default Events
- Generic Database Messages
- Scaffolding
- Customizing Scaffolding
- Customizing the Model-Glue framework

Our Sample Application

- Uses all of the features we will be discussing...

The Include Tag

- Allows you to include a ModelGlue.xml file into another ModelGlue.xml
- They become part of the **same** application
- Include solves a different problem than the ModelGlue_APP_KEY

Include Tag Example

```
<include template="Design/config/ModelGlue.xml" />
```

Include Tag is Useful For....

- Organizing applications into logical subsets
- Creating reusable “sub-applications known as ActionPacks

Include Tag Gotchas

- The last event handler with the same name wins
- You may want to prefix event handlers and message names with a prefix to avoid conflicts:
 - Example.Index, not just Index

Configuring Included Applications

- Use the config tag to set additional viewMappings or beanMappings:

```
<config>
  <setting name="beanMappings"
    value="/Exception/config/ColdSpring.xml" />
  <setting name="viewMappings"
    value="/Exception/views" />
</config>
```

- Used only with ActionPacks

Default Messages

- Model-Glue provides several built in “hooks” so you can do things at certain points during execution.
- Most Model-Glue developers are familiar with default messages:
 - OnRequestStart, OnQueueComplete, OnRequestEnd
- You can create your own message listeners for these messages.
- Provides limited control... what if you could have use an entire event handler?

Default Events

- Model-Glue 2 also provides default events:
 - ModelGlue.OnRequestStart
 - ModelGlue.OnQueueComplete
 - ModelGlue.OnRequestEnd
- You can implement your own event handlers for these events.
 - Can broadcast messages, handle results and include views!
- Only one of each in an entire application.

ModelGlue.OnRequestStart

- Executes before all other event handlers

```
<event-handler name="ModelGlue.OnRequestStart">
  <!-- broadcasts, results and views go here -->
</event-handler>
```

ModelGlue.OnQueueComplete

- Executes after all messages are broadcast but before views are included

```
<event-handler name="ModelGlue.OnQueueComplete">
  <!-- broadcasts, results and views go here -->
</event-handler>
```

ModelGlue.OnRequestEnd

- Executes after all views are included
- Good for a single generic template

```
<event-handler name="ModelGlue.OnRequestEnd">
  <!-- broadcasts, results and views go here -->
</event-handler>
```

Generic Database Messages

- Generic Database Messages provide an easy way to interact with your database:
 - List
 - Read
 - Commit
 - Delete
- Requires the use of an ORM

ModelGlue.GenericList

- Lists records for the specified table
- ```
<message name="ModelGlue.GenericList">
 <argument name="object" value="Customer" />
 <argument name="orderBy" value="LastName" />
</message>
```
- This will run a query of customers ordered by the LastName field.
  - Result is "CustomerQuery" in the event.

## ModelGlue.GenericList Arguments

### Object

- The object type to list

### QueryName (optional)

- The name of the query.
- Defaults to Object & Query (CustomerQuery)

### Criteria (optional)

- A list of values from the event to use as where criteria

### OrderBy (optional)

- A column to order the query by

### Ascending (optional)

- If true, indicates if the order is ascending

## ModelGlue.GenericRead

- Reads a Record for the specified table

```
<message name="ModelGlue.GenericRead">
 <argument name="object" value="Customer" />
 <argument name="criteria" value="customerId" />
</message>
```

- This will load a Record into the event based on the customerId event value.
- Result is "CustomerRecord" in the event.

## ModelGlue.GenericRead Arguments

### Object

- The object type to read

### RecordName (optional)

- The name of the resulting Record.
- Defaults to Object & Record (CustomerRecord)

### Criteria (optional)

- A list of values from the event to use as criteria when loading the Record
- If not provided or not matched a new Record will be created
- You can also specify simple expressions such as deleted=0.

## ModelGlue.GenericCommit

- Commits a Record to the database

```
<message name="ModelGlue.GenericCommit">
 <argument name="object" value="Customer" />
 <argument name="criteria" value="customerId" />
</message>
```

- This will save a CustomerRecord to the database identified by the customerId event value.

## ModelGlue.GenericCommitArguments

### Object

- The object type to commit

### Criteria (optional)

- A list of values from the event to use as criteria when committing the Record
- If not provided or not matched a new Record will be created

### RecordName (optional)

- The name of the resulting Record.
- Defaults to Object & Record (CustomerRecord)

### ValidationName (optional)

- The name of the resulting validation structure.
- Defaults to Object & Validation (CustomerValidation)

### Properties (optional)

- A comma separated list of properties to try to commit from the Event.
- Defaults to all properties in a Record

## ModelGlue.GenericCommitResults

- ModelGlue.GenericCommit will add one of two results. You can map them to events as you want:

### commit

- This is added when the Record is successfully committed.

### validationError

- This is added when there are errors validating the Record before committing it.

## ModelGlue.GenericDelete

- Deletes a Record from the specified table

```
<message name="ModelGlue.GenericDelete">
 <argument name="object" value="Customer" />
 <argument name="criteria" value="customerId" />
</message>
```

- This will delete a Record from the Customer table based on the customerId event value.

## ModelGlue.GenericDeleteArguments

### Object

- The object type to delete

### Criteria (optional)

- A list of values from the event to use as criteria when deleting the Record

## The Scaffold tag

- Defines event handlers for the specified object using database metadata.
- The scaffold tag is an extension of the event handlers tag and can hold all the same children.
- Generates List, View, Edit, Commit, Delete event handlers in the format Object.Event. For example: Customer.List:  
`<scaffold object="Customer" />`
- Useful for generating quick administration interfaces.

## Viewing Scaffolds

- To view a scaffold go to a scaffold event such as Customer.List:
- <http://localhost/index.cfm?event=Customer.List>

## Customize Scaffolded Views

- Scaffolds are generated into a path configured in ColdSpring.xml
  - generatedViewMapping
- This is always the last directory Model-Glue looks in for a view
- Copy the generated view to another view directory and modify it
- Changes to your database will not be reflected any more

## Broadcasts, Results and Views

- Because the `<scaffold>` tag simply is an extension on the `<event-handler>` tag it can hold all of the same child tags.
- Easily allows use of results for templates, etc.
- Easily allows for use of messages for security, etc.

## Example Scaffold

```
<scaffold object="Category">
 <broadcasts>
 <message name="NeedToBeLoggedIn" />
 </broadcasts>
</scaffold>
```

## Scaffold Only What You Want

- Configure Model-Glue to generate only the scaffolds you want by default:  

```
<property name="defaultScaffolds">
 <value>list,edit,view,commit,delete</value>
</property>
```
- Or, use the Scaffold tag type attribute to indicate what should be generated:  

```
<scaffold object="Customer" type="List,View">
```

## Customize Scaffolding

- Model-Glue uses a ColdSpring file to control how scaffolding works:
  - /ModelGlue/unity/config/ScaffoldingConfiguration.xml
- Specify your own Scaffolding Configuration via Model-Glue's ColdSpring configuration:

```
<property name="scaffoldConfigurationPath">
 <value>/Root/config/ScaffoldingConfiguration.xml</value>
</property>
```

## Customize Scaffolding

- In your Scaffolding Configuration you can change how Model-Glue does everything
  - You can specify different XSL files
  - You can define your own scaffold types

## Customize The Model-Glue Framework

- Model-Glue is configured by ColdSpring
  - /modelglue/unity/config/Configuration.xml
- Model-Glue's core configuration is loaded before your ColdSpring.xml
- Settings in your ColdSpring.xml override those in Model-Glue's Configuration.xml
- Allows you to wire in your own versions of Model-Glue core files

## Customize The Model-Glue Framework

- Because you can customize the core framework you can change how parts of the framework behave.
- This can theoretically work for any part of Model-Glue
- May or may not be forward compatible through major versions

## Things You Didn't Learn

- How to integrate Model-Glue with Flex
  - The Flex integration stuff is currently checked into Subversion
- How to announce asynchronous messages
  - Uses Sean Corfield's concurrency library

## What You Learned

- How to use the include tag
- How to use Generic Database Messages
- How to use and customize Scaffolding
- Ways to customize Model-Glue
- Some more areas for study

## Questions and Answers

- Doug Hughes, President
- Alagad Inc
- <http://www.alagad.com>
- [dhughes@alagad.com](mailto:dhughes@alagad.com)
- (888) Alagad4