Fusebox Scaffolder

Version Alpha 3   19 Nov 2009

1. Introduction

The Fusebox Scaffolder is a Fusebox extension, which automatically creates a basic maintenance application for the tables in your database. The application provides facilities to list records from each of the tables, to display the individual records, to add new records, edit existing records and delete records.

The scaffolder is intended to be the first of a number of applications that will provide a facility to generate code automatically, based on a high-level domain specific language.

The Fusebox Scaffolder works by making use of a number of conventions and assumptions about the application so that all you need to do is point the scaffolder at your ColdFusion datasource, choose a template and the application will be built automatically. Once generated you can remove the Fusebox Scaffolder and edit the generated code.

The generated code will probably need to be edited. At this stage the code still isn't 100% reliable so you will almost certainly have to make changes to get the code to run. Over the Beta periods I hope to improve the templates so that they will generate more reliable code, but it is unlikely that it will ever cover every possibility.

You should really think of this tool as a way to avoid 80% of the typing you would normally have to do. The other 20% will be specific to the page layouts you want and any differences in the way the application works.

As part of the process the Fusebox Scaffolder creates an intermediate XML metadata description of your database, called scaffolding.xml by default. This description can be edited before the code build steps are run so that the generated code is more appropriate for your application. This is often the fastest way to make your code more specific.

2. Database Support

Currently only MSSQL 2000, 2005 and 2008 are the only supported DBMSs.

Future versions are planned for Oracle and other databases; however the plan is dependent on volunteers who offer to support each database.

NOTE: Currently the generated code also assumes that the primary key of each table is an integer identity field. Future templates will support primary keys which are integers but not identity with an option to create the Id from a nextId table and UUIDs created by ColdFusion. I would also like some suggestions for other key generation mechanisms that should be supported.

NOTE: It is recommended that the relationships be manually defined in the scaffolder.xml file, as in many cases the database may not contain a complete description of all the relationships. See the description of the XML file below. It is usually much quicker to modify the xml file and regenerate the code than it is to modify the generated code.

3. Conventions

By making use of conventions the configuration required is kept to a minimum. All you have to do is choose a datasource name. Conventions are used to set all the other values required by making assumptions.

The conventions are the ones I use in my applications. Although I follow the Fusebox rules when writing my applications there is a lot of flexibility, so they may be slightly different from your conventions. The main assumptions made are:

1. That each table in the database all has a single column defined as an integer as its primary key and set as identity so that new records are automatically given a unique primary key. If you have any other key arrangements then the scaffolder may not work at all or may generate invalid code.

2. That you want to make use of an MVC style of construction. This may seem overkill for small applications but has been found in practice to be the most flexible way to build Fusebox applications.

3. The controller consists of a circuit.xml file which will be created in a subdirectory of the application called "/controller/*project*/". Where *project* defaults to the name of the selected Datasource.

4. The model consists of a number of CFCs for each table which will be created in a subdirectory of the application called "/model/mDatasource/". If you are using the reactor templates there will be a subdirectory for each type of CFC.

5. The view consists of 3 fuses for each table which will be created in a subdirectory of the application called "/view/v*Project*/".

6. That each type of field is represented by a standard form control, which is set to a standard width. The form control created for each field can be changed easily by changing the intermediate Metadata description file and regenerating.

7. The current code generates an HTML form for editing and adding new records. This could be changed to a Flash form by modifying the templates.

8. The name of each object in the generated code is the name of the table it is generated from, with "tbl" removed from the start if necessary. You can change the generated name in the XML if required.

9. The name of each property in the generated CFCs is the name of the column in the database that stores that property. You can change the property name in the XML if required. (I suspect this may not work well as I haven't tested it)

10. The names of tables and columns are used to generate the label for each object or property. The label is generated by firstly replacing any underscores with a space. If the name is a camel case name a space is added before each capital letter. The first character is capitalised. You can change the generated label in the XML if required.

11. Datetime fields are displayed as a date by default.

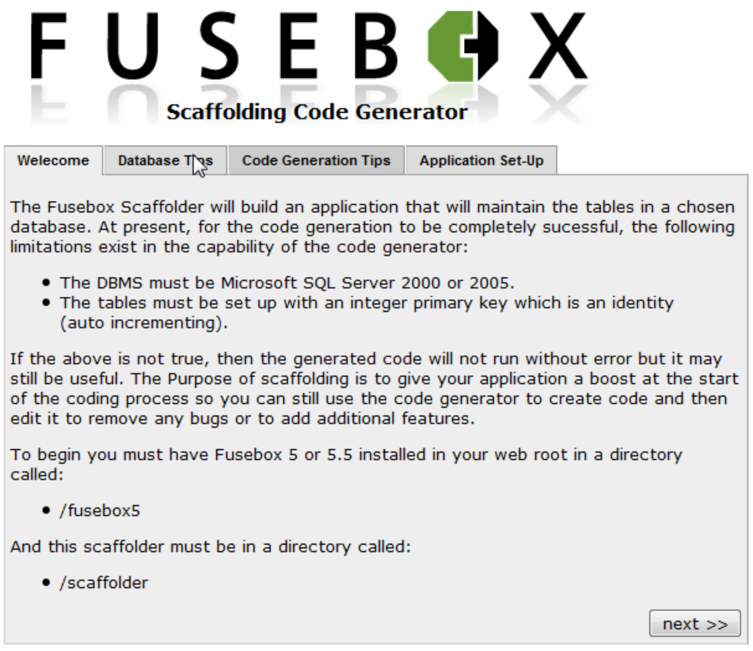
12. Dates are displayed in a format dependant on the machine localisation using LSDateFormat.

13. Unlucky for some, there may be other assumptions. ☺

4. Getting Started

The scaffolder code beta can be downloaded from the Fusebox.org web site and must be placed in a directory called /scaffolder/ in the root of your web server.

The simplest way to use the scaffolder is to make use of the GUI interface. Since the last release a major change has been made to make it much easier to generate your application. You simply open a browser and go to the URL for your server with **http://127.0.0.1/scaffolder/** The scaffolder set-up wizard will display a tabbed interface which offers a number of useful suggestions.



The Fusebox Scaffolder will build an application that will maintain the tables in a chosen database. At present, for the code generation to be completely sucessful, the following limitations exist in the capability of the code generator:

- The DBMS must be Microsoft SQL Server 2000 or 2005.
- The tables must be set up with an integer primary key which is an identity (auto incrementing).

If the above is not true, then the generated code will not run without error but it may still be useful. The Purpose of scaffolding is to give your application a boost at the start of the coding process so you can still use the code generator to create code and then edit it to remove any bugs or to add additional features.

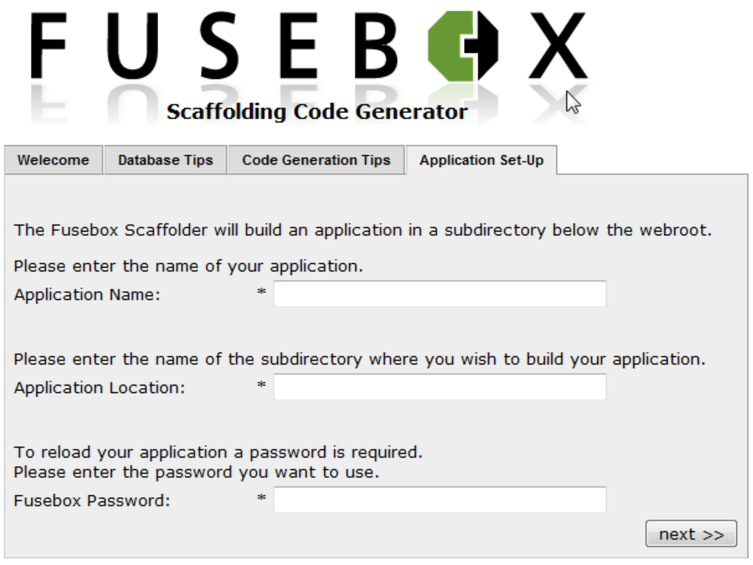
To begin you must have Fusebox 5 or 5.5 installed in your web root in a directory called:

- /fusebox5

And this scaffolder must be in a directory called:

- /scaffolder

Read through the suggestions on each page and on the final page enter the name of your application, the subdirectory you want to place it in and the fusebox password you want your application to use.



The Fusebox Scaffolder will build an application in a subdirectory below the webroot.

Please enter the name of your application.

Application Name: *

Please enter the name of the subdirectory where you wish to build your application.

Application Location: *

To reload your application a password is required.
Please enter the password you want to use.

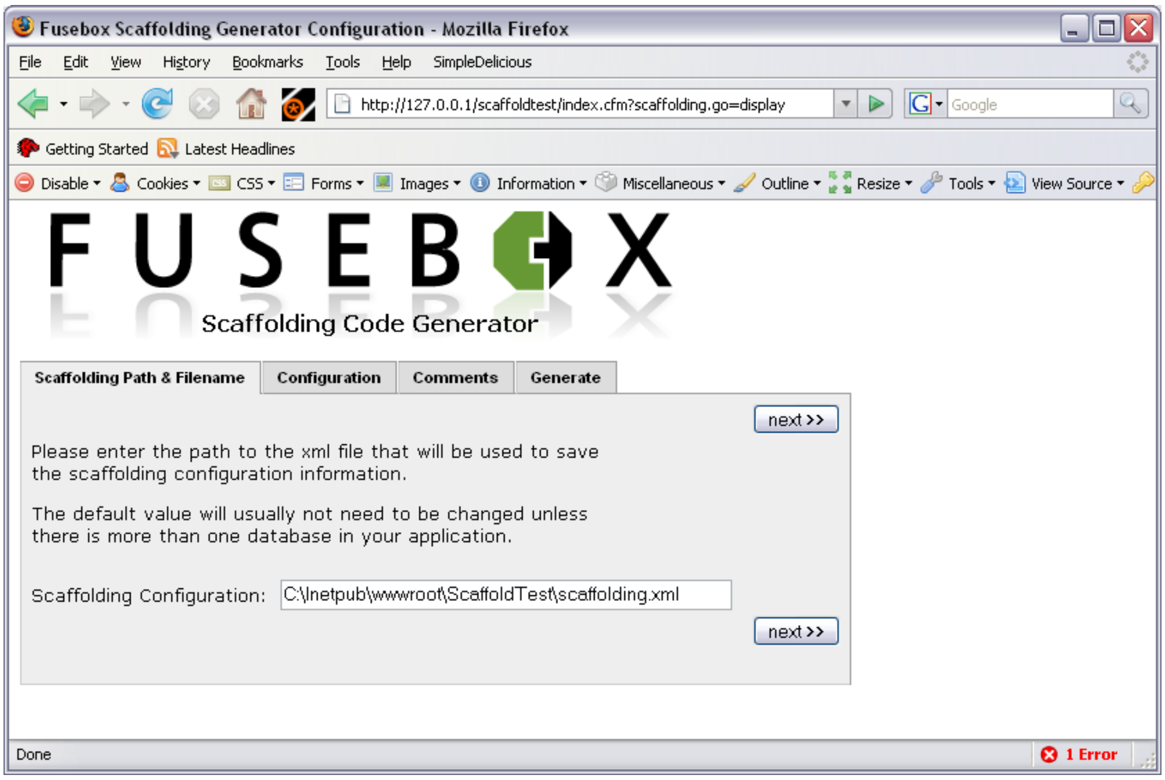
Fusebox Password: *

Click on the **Next>>** button and the GUI will create your application directory and add some initial files.

The GUI will then display a tabbed display page asking for a filename for the intermediate XML file. You can usually just leave this as it is.

By default the Configuration file path will point to a file called scaffolding.xml in the directory you set up for your application. There is usually no need to change this unless your application makes use of multiple databases. If you do need to use multiple databases you should create a separate XML file for each database.

If the XML file does not exist the scaffolder will create one during the Introspection Phase. If it does exist it will be read and used to populate the fields on the subsequent tabs of the form.


Fusebox Scaffolding Generator Configuration - Mozilla Firefox
File Edit View History Bookmarks Tools Help SimpleDelicious
http://127.0.0.1/scaffoldtest/index.cfm?scaffolding.go=display
Getting Started Latest Headlines
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source
FUSEBOX
Scaffolding Code Generator
Scaffolding Path & Filename
Configuration
Comments
Generate
next>>
Done
1 Error

Please enter the path to the xml file that will be used to save the scaffolding configuration information.

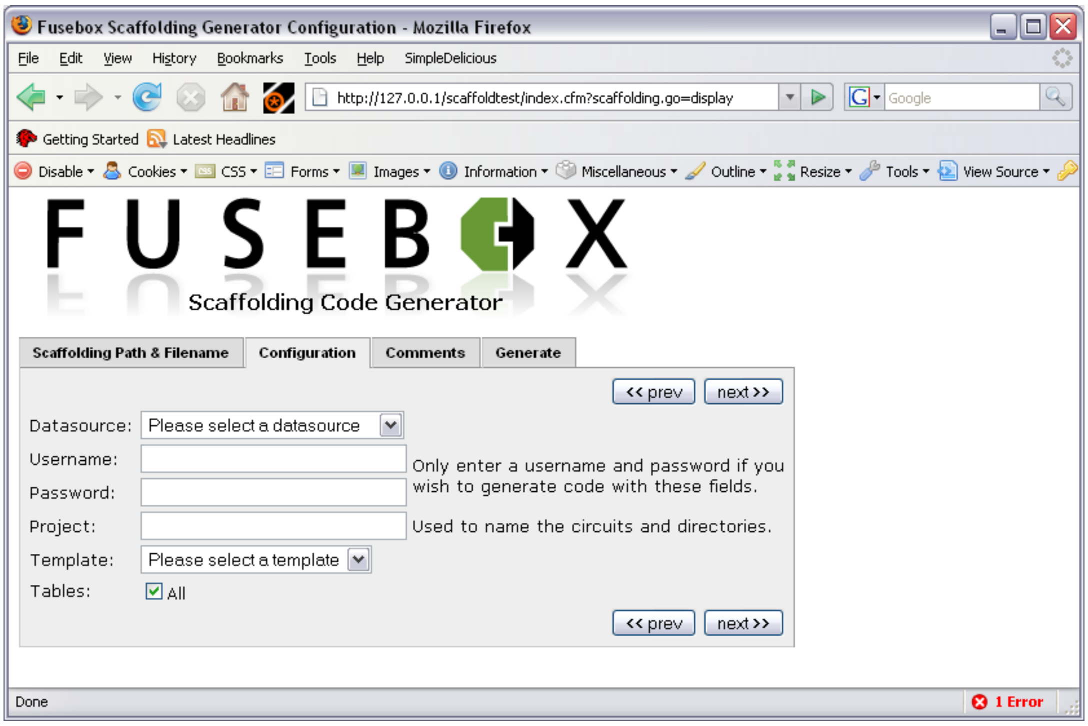
The default value will usually not need to be changed unless there is more than one database in your application.

Scaffolding Configuration: C:\Inetpub\wwwroot\ScaffoldTest\scaffolding.xml

NOTE: At any later time you can display the GUI interface and regenerate the scaffolding files by adding the URL parameter **scaffolding.go=display.**

Click on the **Next>>** button or click on the **Configuration** tab to go to the next tab.

On the configuration tab you can choose the datasource, supply a username and password if required, give the project a name, and chose a template.


Fusebox Scaffolding Generator Configuration - Mozilla Firefox
File Edit View History Bookmarks Tools Help SimpleDelicious
http://127.0.0.1/scaffoldtest/index.cfm?scaffolding.go=display
Getting Started Latest Headlines
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source
FUSEBOX
Scaffolding Code Generator
Scaffolding Path & Filename
Configuration
Comments
Generate
<< prev
next >>
Done
1 Error

Datasource: Please select a datasource

Username:

Password:

Only enter a username and password if you wish to generate code with these fields.

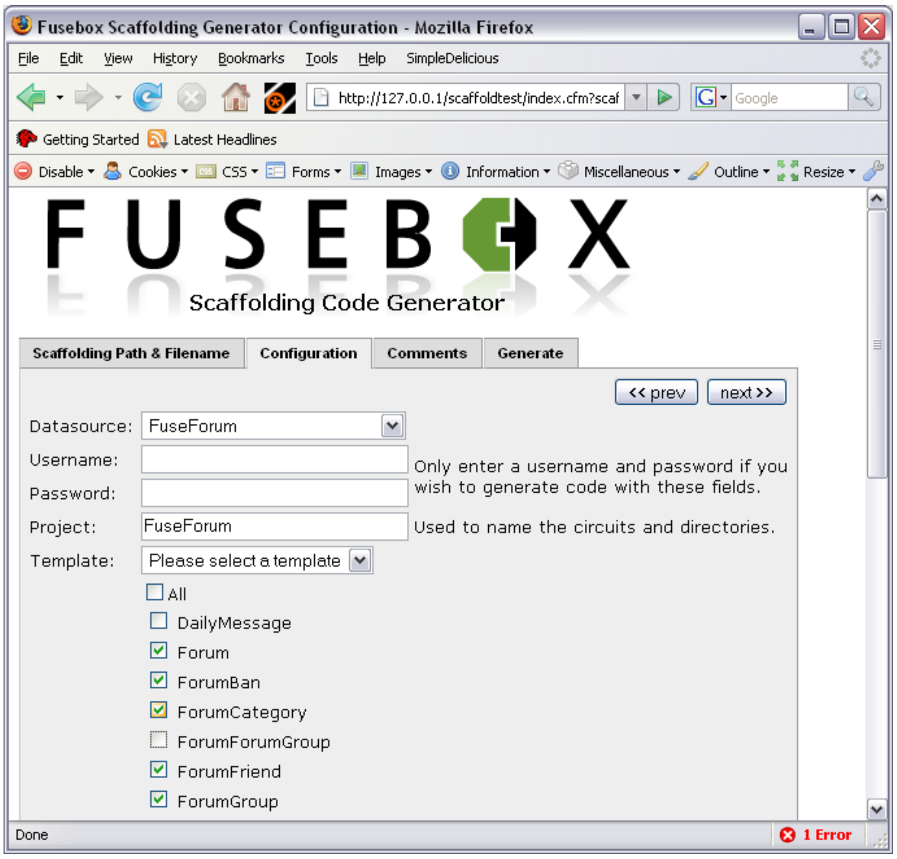
Project:

Used to name the circuits and directories.

Template: Please select a template

Tables: ☑ All

Once the datasource is selected a list of the tables in the attached database will be displayed. If you wish you can choose to omit some of the tables from the generated application.


Fusebox Scaffolding Generator Configuration - Mozilla Firefox
File Edit View History Bookmarks Tools Help SimpleDelicious
http://127.0.0.1/scaffoldtest/index.cfm?scaf
Getting Started Latest Headlines
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize
FUSEBOX
Scaffolding Code Generator
Scaffolding Path & Filename
Configuration
Comments
Generate
<< prev
next >>
Done
1 Error

Datasource: FuseForum

Username:

Password:

Only enter a username and password if you wish to generate code with these fields.

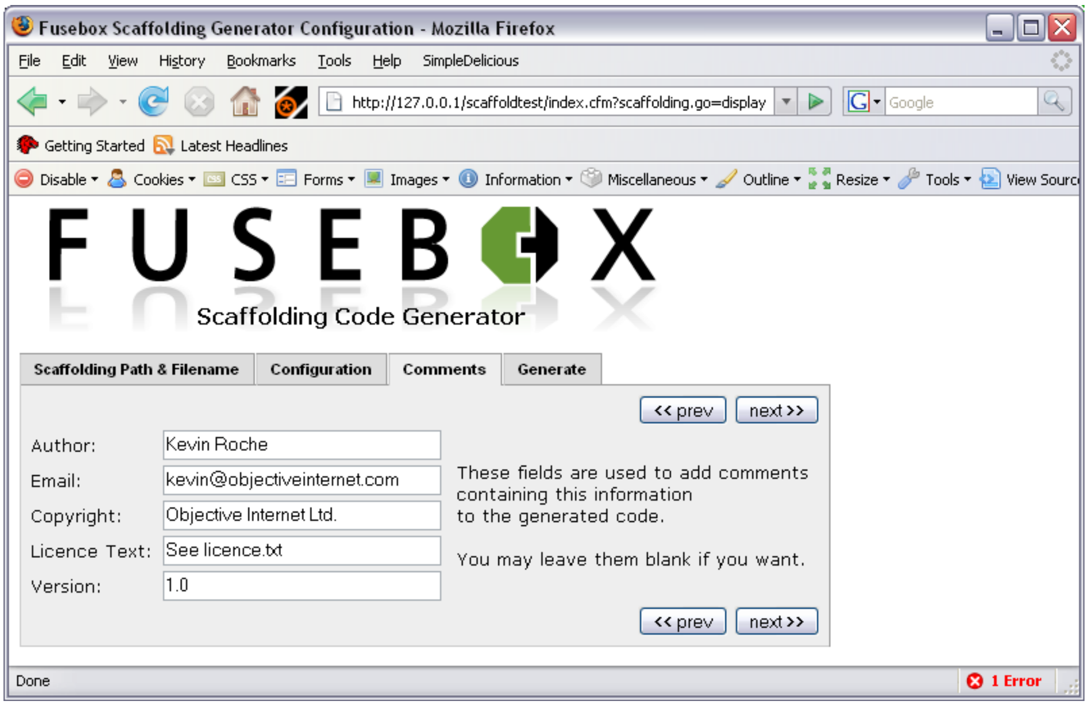
Project: FuseForum

Used to name the circuits and directories.

Template: Please select a template

☐ All
☐ DailyMessage
☑ Forum
☑ ForumBan
☑ ForumCategory
☐ ForumForumGroup
☑ ForumFriend
☑ ForumGroup

Clicking **Next>>** will take you to the **Comments** tab. This tab is used to specify the text that will be used in comments on the application. You can leave all these fields blank if you wish.

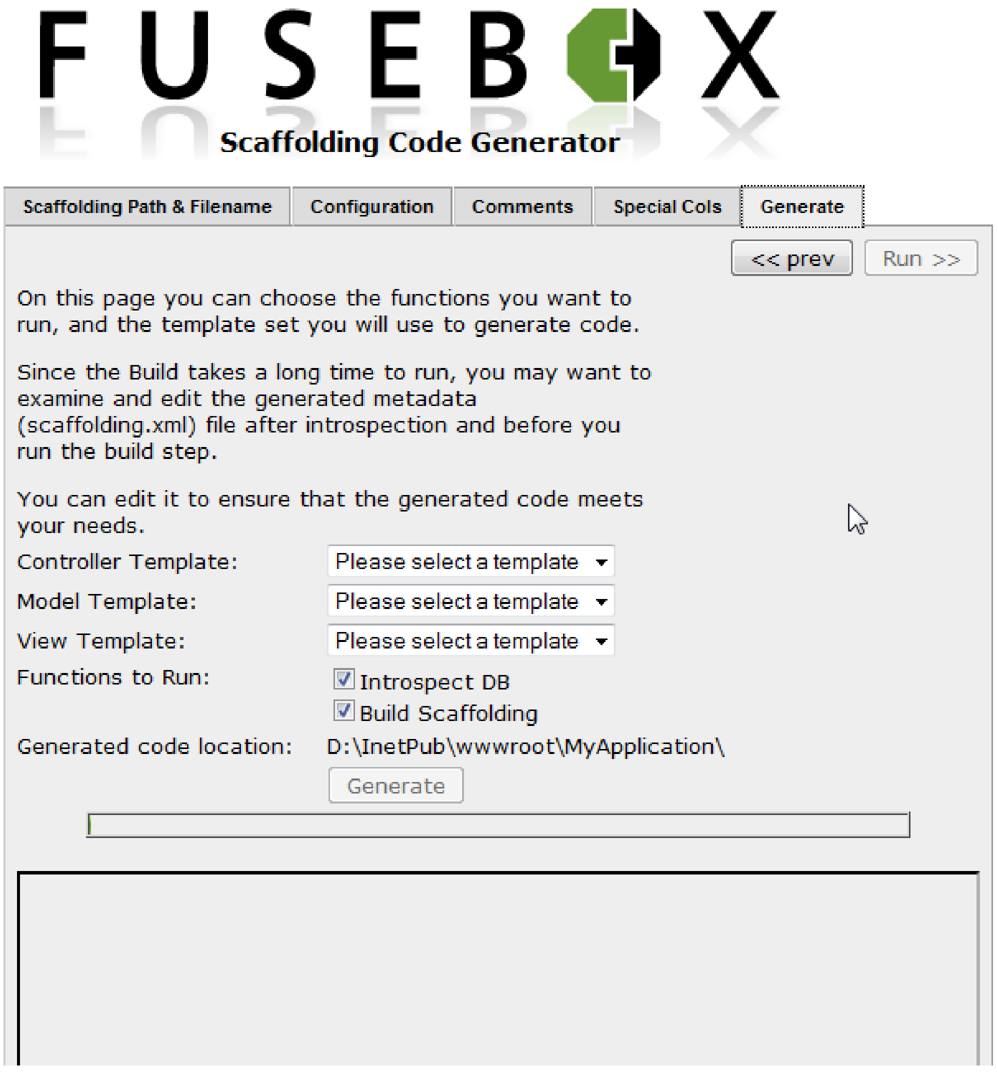

Fusebox Scaffolding Generator Configuration - Mozilla Firefox
File Edit View History Bookmarks Tools Help SimpleDelicious
http://127.0.0.1/scaffoldtest/index.cfm?scaffolding.go=display
FUSEBOX
Scaffolding Code Generator
Scaffolding Path & Filename
Configuration
Comments
Generate
<< prev
next >>
Done
1 Error

Author:	Kevin Roche
Email:	kevin@objectiveinternet.com
Copyright:	Objective Internet Ltd.
Licence Text:	See licence.txt
Version:	1.0

These fields are used to add comments containing this information to the generated code.

You may leave them blank if you want.

Now you can move to the final page which allows you to Introspect the database and generate your scaffolding.



On this page you can choose the functions you want to run, and the template set you will use to generate code.

Since the Build takes a long time to run, you may want to examine and edit the generated metadata (scaffolding.xml) file after introspection and before you run the build step.

You can edit it to ensure that the generated code meets your needs.

Controller Template:	Please select a template
Model Template:	Please select a template
View Template:	Please select a template
Functions to Run:	☑ Introspect DB ☑ Build Scaffolding
Generated code location:	D:\InetPub\wwwroot\MyApplication\

There are three drop down selections of this page which allow you to choose the templates which will be used to generate your controller, model and view.

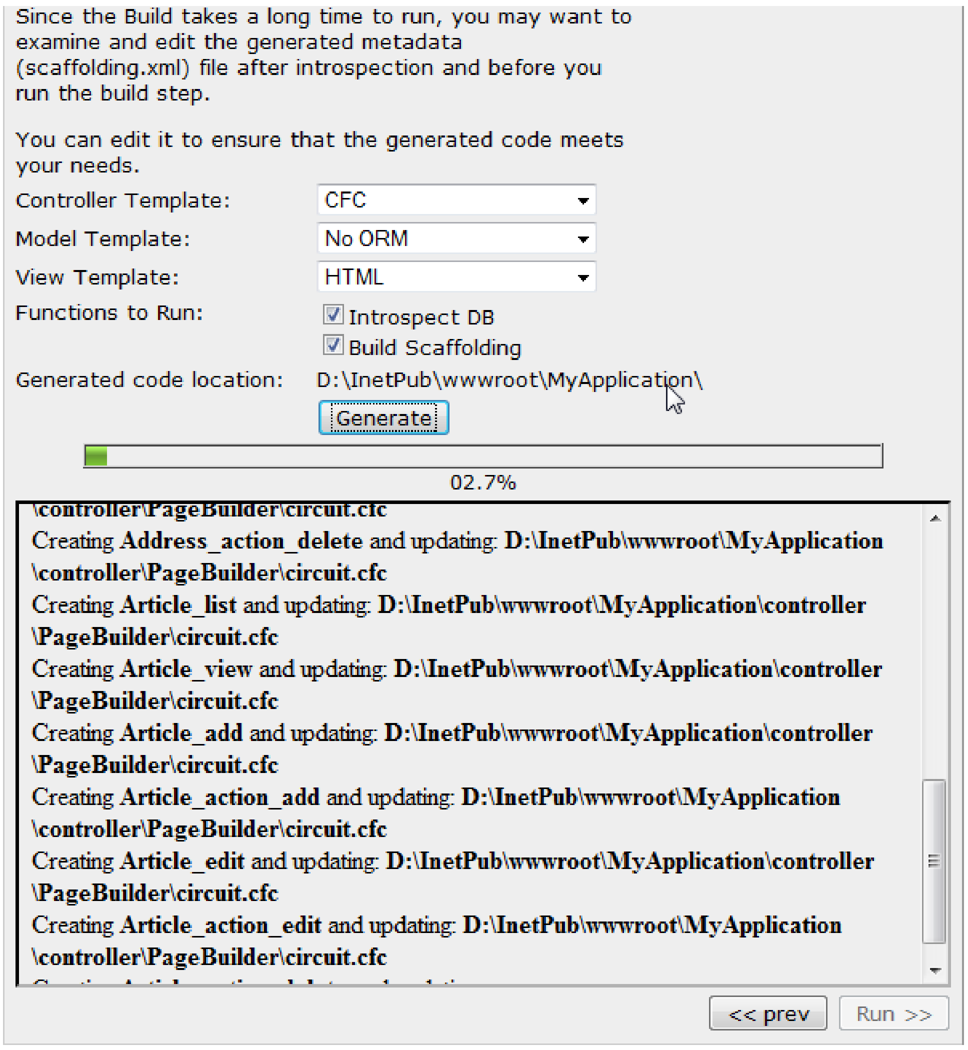
Currently the provided template choices are:

- Controller:
    - XML
    - CFC
- Model:
    - No ORM
- View:
    - HTML

There are also two run options available on this page for Introspect DB and Build Scaffolding, but you must choose to introspect the database the first time you run the scaffolder. This will create the **scaffolding.xml** file. Subsequent times you can decide not to run that step and use the previously generated xml file.

When you click the generate button the scaffolder will begin to generate the application code. Each step will be logged in the window. Overall progress is shown on the progress bar. It may take some time to begin displaying messages in the window as there are no messages generated during introspection.

If you wish you can decide to generate using only one of the three types of template. If so you will get some useful code but not a complete application.



Since the Build takes a long time to run, you may want to examine and edit the generated metadata (scaffolding.xml) file after introspection and before you run the build step.

You can edit it to ensure that the generated code meets your needs.

Controller Template:	CFC
Model Template:	No ORM
View Template:	HTML
Functions to Run:	[✓ Introspect DB] [✓ Build Scaffolding]
Generated code location:	D:\InetPub\wwwroot\MyApplication\

```
\controller\PageBuilder\circuit.cfc
Creating Address_action_delete and updating: D:\InetPub\wwwroot\MyApplication
\controller\PageBuilder\circuit.cfc
Creating Article_list and updating: D:\InetPub\wwwroot\MyApplication\controller
\PageBuilder\circuit.cfc
Creating Article_view and updating: D:\InetPub\wwwroot\MyApplication\controller
\PageBuilder\circuit.cfc
Creating Article_add and updating: D:\InetPub\wwwroot\MyApplication\controller
\PageBuilder\circuit.cfc
Creating Article_action_add and updating: D:\InetPub\wwwroot\MyApplication
\controller\PageBuilder\circuit.cfc
Creating Article_edit and updating: D:\InetPub\wwwroot\MyApplication\controller
\PageBuilder\circuit.cfc
Creating Article_action_edit and updating: D:\InetPub\wwwroot\MyApplication
\controller\PageBuilder\circuit.cfc
```

Once the code has been generated the run button is enabled and you can run the generated application. It will be executed as normal within Fusebox and your new application runs (or crashes!).

5. Customisation

There are several ways to customise the generated application.

1. Adding parameters to the scaffolder calls eg: change the template set to be used.
2. Edit the generated intermediate Meta description XML definition, the scaffolding.xml file, and regenerate the code.
3. Skip the introspection step and provide your own Meta description XML definition.
4. Disable the scaffolder after generating the code and modify the generated code.
5. Modify the templates or create new templates so that the generated code is different. More templates are planned later but there is no reason why you can't make your own which to suit your own programming style or add extra facilities.

5.1 Customising and Reusing the Generated Code

I have tried to make the generated code as flexible as possible so that in many cases it can be easily reused outside of the generated basic maintenance application.

For example:

The standard view files all take a parameter, `variables.fieldlist`, which allows you to change the object properties shown and the order that they are displayed in. The parameter is normally set in the controller's circuit.xml file. You should usually change this in the generated application so that the columns shown on the list page are limited to the most useful ones and fit well on a normal screen.

This also means that you can easily reuse any one of these fuses in a several different contexts, and display a different set of fields in each context.

The code in the standard display fuse is able to show the selected fields from a structure or from the selected row of a query as well as the standard object.

6. Scaffolder Parameters

The parameters to the various scaffolder calls:

Init()

The scaffolder init() method can accept the following parameters:

Parameter	Type	Description
configFilePath	string	Specifies the full file path to the XML configuration file. This XML definition can contain the datasource, username and password as well as additional customisation. If it is not specified the file is created in the application root and named `scaffolding.xml`. More on this later.
datasource	string	Required. The name of a datasource defined in the ColdFusion Administrator.
username	string	Optional. The username to be used when accessing the

		datasource. If omitted the username defined in the ColdFusion Administrator is used.
password	string	Optional. The password to be used with the specified username when accessing the datasource. If omitted the password defined in the ColdFusion Administrator is used.

IntrospectDB()

The introspectDB method can accept :

Parameter	**Type**	**Description**
lTables	list	Optional. Specifies a comma separated list of the names of the tables to be introspected. If missing, all tables in the database are introspected.

build()

The build method can accept :

Parameter	**Type**	**Description**
cftemplate	cftemplate	Required. A reference to a cftemplate object.
template	string	Required. The name of the template set to be used. This points to a subdirectory which contains both the templates and a template descriptor file.
destinationFilePath	string	Optional. Allows you to specify an alternate location for the generated code. If omitted will default to the directory of the index.cfm file which called the scaffolder.
destinationFilePath	string	Optional. Allows you to specify an alternate location for the generated code. If omitted will default to the directory of the index.cfm file which called the scaffolder.
DBName	string	Obsolete. DO NOT USE.
lTables	list	Optional. Specifies a comma separated list of the names of

		the tables for which code is to be generated. If missing, all tables in the database are introspected.

7. Scaffolding XML Metadata file

The scaffolding.xml Metadata file contains a configuration block, which has the details of the datasource to be used, and an objects block, which has details of the objects that are to be maintained and the tables that they relate to.

Here is a sample of part of a typical XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<scaffolding>
	<config>
		<project value="Scaffolding" />
		<dsn value="CMS01AdsTags" />
		<!-- These config values are optional -->
		<username value="myusername" />
		<password value="mypassword" />
	</config>

	<objects>
		<object name="Fuseaction">
			<oneToMany name="Tag" fkname="FK_Fuseaction_Tag">
				<relate from="Fuseaction_Id" to="Fuseaction"/>
			</oneToMany>
			<manyToMany name="TagType">
				<link name="Tag" from="Fuseaction" to="TagType"/>
			</manyToMany>
			<manyToOne name="Site" fkname="FK_Site_Fuseaction">
				<relate from="Site_Id" to="SiteId"/>
			</manyToOne>
			<field alias="Fuseaction_Id"
				formType="Hidden"
				format="Integer"
				identity="true"
				label="Fuseaction Id"
				name="name"
				primaryKeySeq="1"
				required="true"
				showOnForm="true"
				showOnList="true"
				size="0"
				sort="1"
				type="numeric"
				fuseDocType="integer"
				SQLType="CF_SQL_INTEGER"/>
			<field alias="Circuit"
				formType="Text"
				format="Trim"
				label="Circuit"
				maxlength="50"
				name="Circuit"
				required="true"
				showOnForm="true"
				showOnList="true"
				size="30"
				sort="0"
				type="string"
				fuseDocType=" string "
				SQLType="CF_SQL_VARCHAR"/>
```

```
                    <field alias="Site_Id"
                           display="SiteName,SiteURL"
                           formType="Dropdown"
                           format="Trim"
                           label="Site Id"
                           name="Site_Id"
                           parent="Site"
                           required="true"
                           showOnForm="true"
                           showOnList="true"
                           size="1"
                           sort="0"
                           type="numeric"
                           fuseDocType="integer"
                           SQLType="CF_SQL_INTEGER"/>

                    {……more fields}

             </object>

             {……more Objects}

      </objects>

</scaffolding>
```

The xml file config block has a number of elements as follows:

Element	**Description**
project	The name of the project
dsn	The name of the datasource.
username	The DBMS login name (optional)
password	The DBMS password (optional)

NOTE: In the current Alpha 1 version the config block is not updated by introspection.

The objects element will contain an object element for each object to be managed. The object element has the following attributes:

Attribute	**Description**
name	The name of the table that represents this object.
alias	The name it will be known as in your CFML.

Each object element can contain the following elements:

Attribute	**Description**
oneToMany	Describes a one-to-many relationship with another table. The other table is also known as a child table.
manyToOne	Describes a many-to-one relationship with another table. The

	other table is also known as a parent.
manyToMany	Describes a many-to-many relationship with another table.
oneToOne	Describes a one-to-one relationship with another table.
field	Describes a column in the table, which will become a fixed property of the object.
property	Describes an object property that is implemented by a relationship with another table. This allows properties to be implemented that are optional properties and can vary by object instance. **NOTE not yet implemented.**

Within each of the relationship defining tags there is either a set of relate tags or a link tag.

Attribute	**Description**
relate	Used to show which fields in each of the tables relate to each other. Has two parameters: from and to. **from** specifies the alias of the field in the parent table. **to** specifies the alias of the field in the child table.
link	Used to indicate that the relationship is via another table and to define the name of the two relationships involved.

The field tag takes a number of attributes that specify the type of field and the rules for validation and presentation of it.

Attribute	**Description**
alias	The name by which this property will be called in the generated CFML code.
formType	The type of presentation that will be used when showing the field on a form. Valid values in the current Alpha version: • **Dropdown *** • **Radio *** • **Checkbox** • **Textarea** • **Hidden** • **Display** * When Dropdown or Radio is chosen, the generated code expects a query to be available in a parent table with content to populate the list of available choices.
format	The format to be used when displaying the value. Valid values: • **Trim** • **YesNo**

	• **Number(9.99)** where the integers show the display format.
fuseDocType	Indicates the type of field to set up in the generated FuseDocs. Can be one of: • **binary** • **boolean** • **datetime** • **integer** • **number** • **string**
identity	Specifies that the field is an identity generated by the database. Valid values: **true, false**. Default is false, this entity is usually omitted when false.
label	The name that will be used to label the filed on the generated pages.
maxlength	For string fields. This will be used to set the maxlength in form fields. The generated validation code will also limit the length of what can be entered to the specified number of characters.
name	The name of the field in the table.
primaryKeySeq	An integer value. The default is 0. When set to a value greater than 0 this attributes indicates that the field is part of the primary key for the table. Each primary key field is labelled in order of its place in the key.
required	Specifies whether the field is required or not. Valid values: **true, false**.
showOnForm	Specifies whether the field is to be represented on the generated form or not. Valid values: **true, false**.
showOnList	Specifies whether the field is to be represented on the generated list page or not. Valid values: **true, false**.
size	For those form fields that require a size specification. This will specify the width of text boxes and the height of list boxes. On textarea boxes this field is in the format “30x4”. Two integers separated by x, where the first integer is the width and the second is the height.
sort	An integer which when non zero indicates that this field is part of the normal sort sequence used for presenting the table. The value shows the positioning the sort sequence. The sequence is used as the default for list pages and dropdown lists.
SQLtype	Indicates the type of SQL field. Used to set CFSQLType in <cfqueryparam> tags used in the generated code. Can be one of: • **CF_SQL_BIGINT** • **CF_SQL_BIT** • **CF_SQL_CHAR**

	• **CF_SQL_BLOB** • **CF_SQL_CLOB** • **CF_SQL_DATE** • **CF_SQL_DECIMAL** • **CF_SQL_DOUBLE** • **CF_SQL_FLOAT** • **CF_SQL_IDSTAMP** • **CF_SQL_INTEGER** • **CF_SQL_LONGVARCHAR** • **CF_SQL_MONEY** • **CF_SQL_MONEY4** • **CF_SQL_NUMERIC** • **CF_SQL_REAL** • **CF_SQL_REFCURSOR** • **CF_SQL_SMALLINT** • **CF_SQL_TIME** • **CF_SQL_TIMESTAMP** • **CF_SQL_TINYINT** • **CF_SQL_VARCHAR**
type	Indicates the type of field to be used by ColdFusion. Used to set up type rules in the generated code. Can be one of: • **boolean** • **date** • **numeric** • **string**

Notes:

Once the xml has been generated it may be edited manually, if desired, to modify the generated application.

If the xml file already exists when the introspectDB method is run the xml will be updated with any additional fields found in the specified tables. Existing fields and relationships will not be modified, except to add further attributes if necessary.

To ensure that the scaffolder modifies existing fields if they are changed in the database it may be necessary to delete them from the XML file.

It is recommended that the relationships be manually defined, as in many cases the database may not contain a complete description of all the relationships.

TODO: The following section is out of date and needs to be updated:

8. Code Generation

The code is generated from a set of templates that is chosen from one of the available sets. Each set of templates together with a template descriptor is stored in a subdirectory of the templates directory. Currently the provided template choices are:

- Controller:
    - XML
    - CFC
- Model:
    - No ORM
- View:
    - HTML

You can currently choose which of templates are used by setting the parameter when calling the scaffolder code. For example to use the xml templates you would use:

```
oMetaData.build(cftemplate=cftemplate,template="xml");
```

8.1 XML Controller Templates

The following files will be generated from the XML: Metadata description file when using the controller templates:

Filename	**Description**
coldsping.xml	ColdSpring configuration file.
fusebox.xml	Fusebox file.
circuit.xml	Fusebox controller circuit file.
aliasService.cfc	For each table, a complete Service CFC containing methods which create, read, update and delete entries in the table by calling the DAO or Gateway CFCs.
*alias*DAO.cfc	For each table, a complete DAO CFC containing methods which will create, read, update and delete entries in the table.
*alias*Gateway.cfc	For each table, a complete Gateway CFC containing methods which will retrieve sets of records as a query and join the table with any parent tables.
*alias*Record.cfc	For each table, a complete Record CFC containing methods which get and set properties.
aliasTo.cfc	For each table, a complete TransferObject CFC containing properties.
aliasVo.as	For each table, a complete ValueObject containing properties in ActionScript for use with Flash or Flex.

dsp_display_*alias*.cfm	For each table, a Fusebox display fuse for displaying a single record.
dsp_form_*alias*.cfm	For each table, a Fusebox form fuse for adding and editing a single record.
dsp_list_*alias*.cfm	For each table, a Fusebox list fuse for displaying multiple records.

Required Lexicon

In order to make use of the generated code you must also have the ColdSpring lexicon and ColdSpring framework intalled.

Currently the lexicon is delivered as standard with the Fusebox framework extensions distribution.

8.2 CFC Controller Templates

TODO: Create this section.

8.3 No ORM Templates

TODO: Create this section.

8.4 CF9 ORM Templates

NOT YET IMPLEMENTED

8.5 Reactor ORM Templates

NOT YET IMPLEMENTED

The following ColdFusion,Fusebox XML and ORM XML files will be generated from the XML: Metadata description file when using the reactor templates:

Filename	**Description**
reactor.xml	Reactor configuration file (when reactor has been specified as ORM).
fusebox.xml	Fusebox file.
circuit.xml	Fusebox controller circuit file.
circuit.xml	Fusebox model circuit file.
circuit.xml	Fusebox view circuit file.
*alias*Gateway.cfc	For each table, a reactor gateway cfc containing two methods which join the table with any parent tables (when reactor has been specified as ORM).

dsp_display_*alias*.cfm	For each table, a Fusebox display fuse for displaying a single record.
dsp_form_*alias*.cfm	For each table, a Fusebox form fuse for adding and editing a single record.
dsp_list_*alias*.cfm	For each table, a Fusebox list fuse for displaying multiple records.

Required Lexicon

In order to make use of the generated code you must also have the Reactor ORM lexicon and the Reactor Object Relational Mapping framework inatlled.

Currently the lexicon is delivered as standard with the Fusebox framework extensions distribution.

8.6 Transfer ORM templates

NOT YET IMPLEMENTED

The following ColdFusion, Fusebox XML and ORM XML files will be generated from the XML: Metadata description file when using the ColdSpring templates:

Filename	**Description**
datasource.xml	Transfer datasource configuration file (when transfer has been specified as ORM). **Not Yet Implemented.**
transfer.xml	Transfer configuration file (when transfer has been specified as ORM). **Not Yet Implemented.**
fusebox.xml	Fusebox file.
circuit.xml	Fusebox controller circuit file.
*alias*Gateway.cfc	For each table, a reactor gateway cfc containing two methods which join the table with any parent tables (when reactor has been specified as ORM).
dsp_display_*alias*.cfm	For each table, a Fusebox display fuse for displaying a single record.
dsp_form_*alias*.cfm	For each table, a Fusebox form fuse for adding and editing a single record.
dsp_list_*alias*.cfm	For each table, a Fusebox list fuse for displaying multiple records.

8.7 HTML View Templates

TODO: Create this section.

8.8 Ext JS View Templates

TODO: Create this section.

9. Creating your own templates

There is nothing to stop you creating your own templates and using them to generate your code. The template format is based on Peter Bell's cftemplate and uses coldfusion code as the template language. You can read more about this on Peter's blog at www.pbell.com.

9.1 *Location*

Templates are stored in subdirectories of the templates directory. Eg:

- scaffolder/templates/model/XML
- scaffolder/templates/view/HTML

The easiest way to create your own is to copy an existing set of templates to a new subdirectory. Eg:

- scaffolder/templates/view/mytemplates

9.2 *Template File Format*

A template contains some CFML or other code that will be used to create the final generated code. It also uses a modified version of CFML to call the metadata object to get information and modify the code.

The CFML code that modifies the template is written with double brackets (**<<cfoutput>>)** around the tags and comments and double dollar signs around variables (**$$variable$$**).

Here is a very simple example template:

```
<<!--- Set the name of the object (table) being updated --->>
<<cfset objectName = oMetaData.getSelectedTableAlias()>>

<<cfoutput>>
	<cffunction name="save" access="public" output="false"
returntype="boolean">
		<cfargument name="$$objectName$$" type="$$objectName$$"
required="true" />

		<cfset var success = false />
		<cfif exists(arguments.$$objectName$$)>
			<cfset success = update(arguments.$$objectName$$) />
		<cfelse>
			<cfset success = create(arguments.$$objectName$$) />
		</cfif>

		<cfreturn success />
	</cffunction>
<</cfoutput>>
```

In the simple example most of the code here looks like normal CFML with a couple of exceptions:

- Some double tags are used (eg: **<<cfoutput>>**) to execute code at generate time.
- Some double dollar signs are used (eg **$$variable$$**) to get the values of variables at generate time.

The CFML code written this way gets executed during the code generation phase any normal CF code is simply output when the template is used to generate code.

The most common thing we want to do in a template is loop over a list of fields from a table. You can do this with something like:

```
<<!--- Generate a list of the Primary Key fields --->>
<<cfset lPKFields = oMetaData.getPKListFromXML(objectName)>>

<<!--- Loop over a list of the Primary Key fields and create a cfset--->>
<<cfloop list="$$lPKFields$$" index="thisPKField">>
 <set name="attributes._listSortByFieldList"
        value="$$objectName$$|$$thisPKField$$|ASC" overwrite="false" />
<</cfloop>>
```

Other commonly used calls to oMetadata will retrieve useful values which can be used to populate the template.

```
<<!--- Set the name of the object (table) being updated --->>
<<cfset objectName = oMetaData.getSelectedTableAlias()>>

<<!--- Generate a list of the Primary Key fields --->>
<<cfset lPKFields = oMetaData.getPKListFromXML(objectName)>>

<<!--- Generate a list of the table fields --->>
<<cfset lFields = oMetaData.getFieldListFromXML(objectName)>>

<<!--- Generate a list of the joined fields --->>
<<cfset lJoinedFields = oMetaData.getJoinedFieldListFromXML(objectName)>>
<<cfset lAllFields = ListAppend(lFields,lJoinedFields)>>

<<!--- Generate a list of the Primary Key fields --->>
<<cfset lPKFields = oMetaData.getPKListFromXML(objectName)>>
```

There are two calls to oMetadata which return arrays of structures. The first gets data about the current table and the second about tables which have a parent type (many to one) relationship with the current table.

```
<<!--- Get an array of fields --->>
<<cfset aFields = oMetaData.getFieldsFromXML(objectName)>>

<<!--- Create a array of the many to one joined objects --->>
<<cfset aJoinedObjects =
 oMetaData.getRelationshipsFromXML(objectName,"manyToOne")>>
```

Both of these arrays contain structures with the complete metadata entry for each of the fields. You can then get at the individual values by using something like: $$aFields[i].name$$.

```
<<cfloop from="1" to="$$ArrayLen(aFields)$$" index="i">>
 $$objectName$$.$$aFields[i].name$$ AS $$aFields[i].alias$$
<</cfloop>>
```

9.3 Template Descriptor File

Once you have created a template you have to tell the scaffolding about it and how it should be used. To do this a single template descriptor file is used. In the current version the template descriptor file is a cfscript based file which populates an array of structures with one entry for each template.

For example:

```
stFileData = structNew();
stFileData.templateFile = "controller_circuit.xml";
stFileData.outputFile = "circuit.xml";
stFileData.MVCpath = "#destinationFilePath#controller\#variables.datasource#\";
stFileData.inPlace = "false";
stFileData.overwrite = "false";
stFileData.useAliasInName = "false";
stFileData.suffix = "cfm";
stFileData.perObject = "false";
ArrayAppend(aTemplateFiles,stFileData);
```

The above code is an example of a single entry in the file, which specifies the details for a single template. The template descriptor file is stored in the same directory as the templates and is always called **templateDescriptor.cfm**.

Each of the entries in the array must have the following entries in the structure. The entries have the following meanings:

Variable	**Description**
templateFile	Specifies the name of the template file without any suffix.
outputFile	Specifies the name of the file to be generated without any suffix.
MVCpath	Full path to the directory which will contain the generated file.
inPlace	Specifies if the template generates the whole output file or just part of it. If true the template will generate a small part of the file which will be inserted or used to update the appropriate part of the file. Valid values are **true** and **false**.
overwrite	Specifies if the file or part file should be overwritten if it already exists. Valid values are **true** and **false**.
useAliasInName	Specifies if the filename should be changed to include the object alias to make it unique. Files with a file suffix of **.cfc** or **.as** will use the object name as a prefix, other files will use it as a suffix to the main part of the name. Eg: **DailyMessageDAO.cfc** or **dsp_display_DailyMessage.cfm** Valid values are **true** and **false**.
suffix	Specifies the file suffix for both template and generated file.
perObject	Should a separate copy be generated for each object or only a single one generated for the application. Valid values are **true** and **false**.