

Monstarillo Template Modification Guide

Overview.....	6
Template Generation Process	8
How Template Sets are Defined	8
TemplateFile	8
TemplateSet	9
TemplateSetFiles.....	9
GUI Creation.....	11
Custom Templates	13
Custom Tag Types	13
Per Column Tags.....	13
Per Table Tags	13
All Table Tags.....	14
How to Create a Custom Tag.....	14
MonstarilloDBObject.....	15
Using Monstarillo DB Object.....	15
MonstarilloDBSample project	16
Tutorials	18
Tutorial 1 Creating Your First Template.....	18
Tutorial 2 Creating a Custom Tag	24
Tag Reference	29
ForEach Tags	29
ForEachColumn	29
ForEachColumnRemoveComma	29
ForEachNullableColumn	29
ForEachNonNullableColumn	29
ForEachIdentityColumn.....	29
ForEachIdentityColumnRemoveComma.....	29
ForEachNonIdentityUpdatableColumnRemoveComma	30
ForEachNonIdentityUpdatableColumn	30
ForEachNonIdentityColumn.....	30
ForEachNonIdentityColumnRemoveComma.....	30
ForEachTextColumn.....	30
ForEachImageColumn	30
ForEachPrimaryIdentityColumn.....	30
ForEachPrimaryNonIdentityColumn.....	30
ForEachUpdatableColumn.....	30
ForEachUpdatableColumnRemoveComma.....	31
ForEachUpdatableNonNullableNonIdentityColumn.....	31
ForEachPrimaryColumn	31
ForEachPrimaryColumnRemoveComma	31
ForEachNonPrimaryColumn	31
ForEachNonPrimaryColumnRemoveComma	31
ForEachPrimaryNonIdentityColumnRemoveComma	31
ForEachForeignKeyPKOnly.....	32

ForEachForeignKeyFKOnly.....	32
Column Tags.....	32
ColumnName.....	32
DBColumnName.....	32
ColumnSQLType.....	32
ColumnTSQLType.....	32
Length.....	32
SqlCodeType.....	32
NetCodeType.....	33
SqlCastType.....	33
TableName.....	33
DBTableName.....	33
NumericPrecision.....	33
NumericScale.....	33
IsNullable.....	33
ColumnDisplayName.....	33
Stored Procedure Tags.....	33
SqlCodeType.....	33
SqlCastType.....	34
SprocParameterList.....	34
SprocNameCode.....	34
SprocName.....	34
ParameterName.....	34
ParameterCodeName.....	34
NumericScale.....	34
NumericPrecision.....	34
NetCodeType.....	34
Length.....	35
ColumnTSQLType.....	35
ColumnSQLType.....	35
Foreign Key Tags.....	35
FK_PrimaryKeyTable.....	35
FK_PrimaryKeyTableDb.....	35
FK_PrimaryKeyColumn.....	35
FK_PrimaryKeyColumnDb.....	35
FK_ForeignKeyTable.....	35
FK_ForeignKeyTableDb.....	36
FK_ForeignKeyColumn.....	36
FK_ForeignKeyColumnDb.....	36
FK_PrimaryKeyColumnType.....	36
FK_ForeignKeyColumnType.....	36
FK_SqlBefore.....	36
FK_SqlAfter.....	36
Global Tags.....	36
AccessModifier.....	36
AllColumnsList.....	37

AllColumnsListIdentityOutput	37
AllColumnsListWithNativeTypes	37
AllColumnsListWithNativeTypesIdentityOutput	37
AllColumnsListWithTypes	37
AllColumnsListWithTypesIdentityOutput	37
AllUpdatableColumnsList	37
AllUpdatableColumnsListIdentityOutput	38
AllUpdatableColumnsListWithNativeTypes	38
AllUpdatableColumnsListWithNativeTypesIdentityOutput	38
AllUpdatableColumnsListWithTypes	38
AllUpdatableColumnsListWithTypesIdentityOutput	38
ConnectionString	38
Database	38
DBTableName	38
DBVersionIdentityReturn	39
DeveloperName	39
FillDropDowns	39
FillDropDownsNativeTypes	39
ForeignKeyIndexers	39
ForeignKeyIndexersMSAppBlock	39
ForeignKeyIndexersNativeTypes	39
IdentityColumnsListWithTypes	39
InsertParamColumnCount	40
InsertParamColumnCountPlus1	40
Namespace	40
NonIdentityColumnsListWithTypes	40
NToNIndexersBusiness	40
NToNIndexersBusinessNativeTypes	40
NToNIndexersDABBusiness	40
NToNIndexersDABPersist	40
NToNIndexersDABStoredProc	40
NToNIndexersInlinePersist	41
NToNIndexersPersist	41
NToNIndexersPersistNativeTypes	41
NToNIndexersStoredProc	41
PrimaryColumnCount	41
PrimaryColumnCountPlus1	41
PrimaryKeyList	41
PrimaryKeyListForQueryString	41
PrimaryKeyListWithSqlTypes	41
SprocCodeName	42
SprocName	42
SprocParameterList	42
SprocPrefix	42
TableName	42
UpdatableParamColumnCount	42

UpdatableParamColumnCountPlus1 42

Overview

Monstarillo is a template based code generator. This guide will teach you all you need to know to modify existing templates and create your own.

Monstarillo templates are tag based. Monstarillo tags are text surrounded by “%%”. The templates that ship with Monstarillo can be found in the \Program Files\Yellow Bridge Software Inc\Monstarillo\Templates directory.

There are seven different types of Monstarillo Tags.

1. **ForEach Tag:** A tag that begins a loop of some sort. It may loop through all of the columns in a table, all of the primary key columns of a table or all of the identity columns of a table. There are over 20 ForEach tags in Monstarillo. ForEach tags must be followed by a %%EndFor%% tag. ForEachColumn, ForEachIdentityColumn, and ForEachNonIdentityColumn are examples of ForEach tags. Users cannot create ForEach tag but can create their own per table tags which will accomplish the same thing.
2. **Table/View Column Tag:** A tag that represents a property of a column. ColumnName, TableName, NumericPrecision and IsNullable are examples of column tags. Users can create their own column tags.
3. **Stored Procedure Tag:** A tag that represents a property of a stored procedure parameter. ParameterName, SPROCName and ColumnSQLType are examples of stored procedure tags.
4. **Foreign Key Tag:** A tag that represents a property of a given tables foreign keys. FK_PrimaryKeyTable, FK_PrimaryKeyColumn, FK_ForeignKeyTable and FK_ForeignKeyColumn are examples of foreign key tags.
5. **Global Tag:** Global tags are run once per entity. When running Monstarillo against tables and views global tags are run once per table/view. When running Monstarillo against stored procedures global tags are run one per stored procedure. Global tags have access to all of the entities being processed. Monstarillo uses global tags to generate code for foreign key indexers, N:N Indexers. Global tags are used to hold the user preferences like Base Namespace, DeveloperName and stored procedure prefix. Global tags are also used to create strings that are used as parameters in code like a list of all primary key columns or updatable columns. ForeignKeyIndexers, Namespace, DeveloperName, PrimaryKeyList and PrimaryKeyListWithTypes are examples of global tags. Users can create their own global tags.
6. **PreProcess Tags:** Monstarillo gives the template writer the ability to generate GUI code for different controls using the same tag. For example the code to fill a TextBox and a CheckBox are different but can be accomplished in a Monstarillo template via the use of a PreProcess Tag. PreProcess tags are defined in the CreateGUI.xml file in your templates

directory. This file tells Monstarillo what to do for each of the control types when it is encountered in a template. The CreateGUI.xml file is described in detail later in this guide.

7. **Template Type Tags:** By default Monstarillo will run each template for every table and view selected by the user at runtime. In some cases you will want a given template to only run against tables or only views. To instruct Monstarillo to run a given template against tables only, make the following line of code the first line of code in your template `<TemplateType>TableOnly</TemplateType>`. To instruct Monstarillo to run a given template against views only make the following line of code the first line of code in your template `<TemplateType>ViewOnly</TemplateType>`. Template Type tags must start at the first character of the first line of a template.

Template Generation Process

Monstarillo generates code against a template set. A template set is a group of one or more template files. Template sets and template files are defined in the Templates.xml file. Monstarillo requires the user to select a template path before selecting a template set to run. The templates.xml file in the folder of the selected path is used by Monstarillo. Templates.xml is a serialized strongly typed DataSet. When modifying templates it is recommended that you make a copy of the templates that ship with Monstarillo before modifying them. A GUICreate.xml file is also required by Monstarillo. This file is explained later in this guide. Figure 1 is a representation of the templates.xml schema.

Figure 1 shows three screenshots of XML schema tables. The first screenshot shows the 'TemplateFile' table with columns: FileID (int), FileName (string), FileExtension (string), NewFileName (string), FolderName (string), OverWriteExisting (string), AccessModifier (string), and Append (string). The second screenshot shows the 'TemplateSetFiles' table with columns: TemplateSetName (string) and FileID (int). The third screenshot shows the 'TemplateSet' table with columns: TemplateSetName (string), TemplateSetDescription (string), Database (string), Language (string), NeedsGui (string), and Order (string).

Figure 1

How Template Sets are Defined

TemplateFile

A TemplateFile describes a Monstarillo template. In most cases Monstarillo will generate a file for each entity Monstarillo is run against. If a user were to run Monstarillo against ten tables, ten files would be generated for each TemplateFile in the TemplateSet run. The template file tells Monstarillo what to name each generated file and what folder to put each generated file in.

Following is a description of each of the columns in a template file.

FileID: a unique number to define a template file.

FileName: The name of the template file. The actual template file must be located in the same directory as the templates.xml file.

FileExtension: The extension of the generated file.

NewFileName: The name of the generated file. The string “TableName” in this column will be replaced by the name of the table or view that Monstarillo is run against.

FolderName: The name of the folder to place the generated files into. The folder will be created if it does not already exist. The folder will be located under the Output Path that the user selects in the Monstarillo GUI when the template set is run. If this column is left blank the files will be put in the Output Path that the user selected.

OverWriteExisting: If set to Y Monstarillo will overwrite a previously generated file if it exists. If set to N Monstarillo will not overwrite a previously generated file if it exists.

AccessModifier: The value in this column will be returned by the %%AccessModifier%% tag.

Append: Monstarillo generates a file per entity unless this property is set to T. When set to T the generated file is appended to for each entity. This file is deleted at the beginning of each Monstarillo run if it exists.

TemplateSet

TemplateSets define a template set. Following is a description of each TemplateSet column.

TemplateName: The name of the template set. This is the name of the template set that is displayed in Monstarillo.

TemplateSetDescription: The description of the template set. This is the description of the template set displayed in Monstarillo.

Database: The database the template is designed for. Monstarillo currently supports Microsoft Sql Server.

Language: The language the template creates.

NeedsGui: Not currently used. It remains for backwards compatibility.

Order: The order of the template sets as displayed in the Monstarillo GUI.

TemplateSetFiles

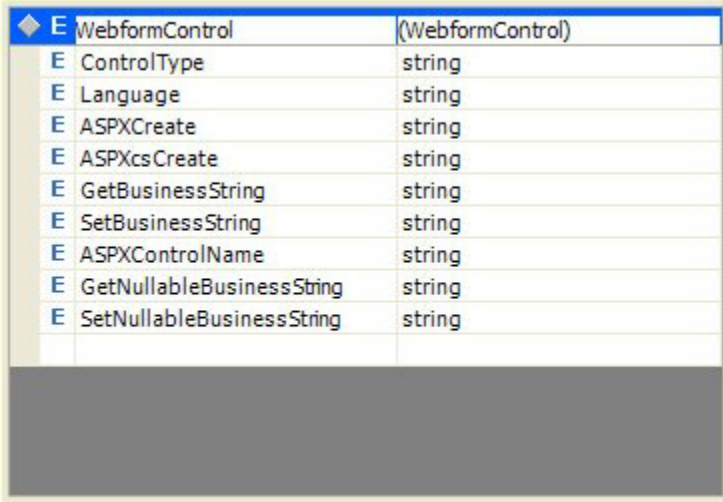
TemplateSetFiles is what joins a TemplateSet and a TemplateFile.

TemplateName: The name of the TemplateSet to add a TemplateFile to.

FileID: The FileID of the TemplateFile to add to the TemplateSet.

GUI Creation

Monstarillo gives the template writer the ability to generate code for the different types of controls it supports with the same tag by using the GUICreate.xml file located in the templates directory. Monstarillo allows a user to define which type of control it wants to use to display data for a given column. Monstarillo allows a user to customize the code it generates when a control is created in an aspx and aspx.cs file, when a property is set or retrieved in a business object, and the name of the control. This information is stored in the CreateGUI.xml file. CreateGUI.xml is a serialized strongly typed DataSet Figure 2 is a representation of the CreateGUI.xml schema.



E WebformControl (WebformControl)		
E	ControlType	string
E	Language	string
E	ASPXCreate	string
E	ASPXcsCreate	string
E	GetBusinessString	string
E	SetBusinessString	string
E	ASPXControlName	string
E	GetNullableBusinessString	string
E	SetNullableBusinessString	string

Figure 2

Following is a description of each of the columns. GetNullableBusinessString and GetBusinessString do the same thing. Monstarillo uses the GetBusinessString column in templates that use SqlTypes and GetNullableBusinessString in templates that use native types.

ControlType: The type of control being defined

Language: The language the code generates.

ASPXCreate: The code used to create the control in an aspx page.

ASPXcsCreate: The code used to create the control in an aspx.cs page.

GetBusinessString: The code used to set a property on a Business object.

SetBusinessString: The code used to retrieve a property from a Business object.

GetNullableBusinessString: The code used to set a property on a Business object.

SetNullableBusinessString: The code used to retrieve a property from a Business object.

Custom Templates

Custom templates gives a template writer the ability to further customize the code generation. Monstarillo's plug in architecture gives you access to the entire Monstarillo code generation engine.

Custom Tag Types

There are three different types of Monstarillo custom tags. The different tag types differ in how often they are called for each entity in a code generation run and which Monstarillo objects they have access to.

Per Column Tags

A **per column tag** is called once per column and will give you access to the Monstarillo column that is currently being processed. Per column tags must be inside of a ForEach tag. To create a per column tag you will create a class that implements the `IMonstarilloPerColumnTag` interface. Notice that the `PerformAction` method has an `IMonstarilloPerColumnContext` as a parameter. The interface definitions are below.

```
public interface IMonstarilloPerColumnTag
{
    string Name{get;}
    void PerformAction( IMonstarilloPerColumnContext context );
}

public interface IMonstarilloPerColumnContext
{
    Column MonstarilloColumn{get;set;}
    String Code{get;set;}
}
```

Per Table Tags

A **per table tag** is called once per table and will give you access to the Monstarillo Table that is currently being processed. To create a per table tag you will create a class that implements the `IMonstarilloPerTableTag` interface. Notice that the `PerformAction` method has an `IMonstarilloPerTableContext` as a parameter. The interface definitions are below.

```
public interface IMonstarilloPerTableTag
{
    string Name{get;}
    void PerformAction( IMonstarilloPerTableContext context );
}

public interface IMonstarilloPerTableContext
```

```

{
    Table MonstarilloTable{get;set;}
    string Code{get;set;}
}

```

All Table Tags

A **all tables tag** is called once per table and will give you access to all of the Monstarillo tables. To create a all tables tag you will create a class that implements the `IMonstarilloAllTablesTag` interface. Notice that the `PerformAction` method has an `IMonstarilloAllTablesContext` as a parameter. The interface definitions are below.

```

public interface IMonstarilloAllTablesTag
{
    string Name{get;}
    void PerformAction( IMonstarilloAllTablesContext context );
}

public interface IMonstarilloAllTablesContext
{
    TableCollection MonstarilloTableCollection{get;set;}
    string CurrentMonstarilloTableName{get;set;}
    string Code{get;set;}
}

```

How to Create a Custom Tag

There are three steps to creating a custom tag.

Step 1

Create a class that implements `IMonstarilloPerTableTag`, `IMonstarilloPerColumnTag`, or `IMonstarilloAllTablesTag` depending on the type of tag you wish to create.

Step 2

Place the assembly in the same folder as `Monstarillo.exe`.

Step 3

Modify the `PlugIns.xml` file located in the same folder as `Monstarillo.exe`.

The schema for the `PlugIns.xml` file is pictured below

Class Name	Property Name	Property Type
PerColumnTag (PerColumnTag)	Tag	string
	AssemblyName	string
	ClassName	string
PerTableTag (PerTableTag)	Tag	string
	AssemblyName	string
	ClassName	string
AllTablesTag (AllTablesTag)	Tag	string
	AssemblyName	string
	ClassName	string

MonstarilloDBObject

A MonstarilloDBObject is an object that defines all of the tables, views and stored procedures you have selected for generation in Monstarillo.

MonstarilloDBObjects are created by running the MonstarilloDBObject template set in Monstarillo.

Running the MonstarilloDBObject will produce two files. <database name>.bin and <databasename>Sprocs.bin. The <database name>.bin is a collection of Table objects that have been serialized using a binary formatter. The <database name>Sprocs.bin is a collection of StoredProcedure objects that have been serialized using a binary formatter. The Table and StoredProcedure objects are fully documented in the MonstarilloTypes documentation which can be found in MonstarilloTypesDocumentation.chm located in the Monstarillo home directory.

Using Monstarillo DB Object

Run Monstarillo and create a MonstarilloDBObject for your database.

Create a new Project in Visual Studio.

Add a reference to MonstarilloTypes.dll

Add the following using statements to your class:

```
using System.Data;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
```

Instantiate the MonstarilloDBObject. The following code instantiates a MonstarilloDBObject that has been saved to a file called "Northwind.bin" located in the same directory as my executable.

```
FileStream fs = new FileStream("Northwind.bin", FileMode.Open);

BinaryFormatter formatter = new BinaryFormatter();

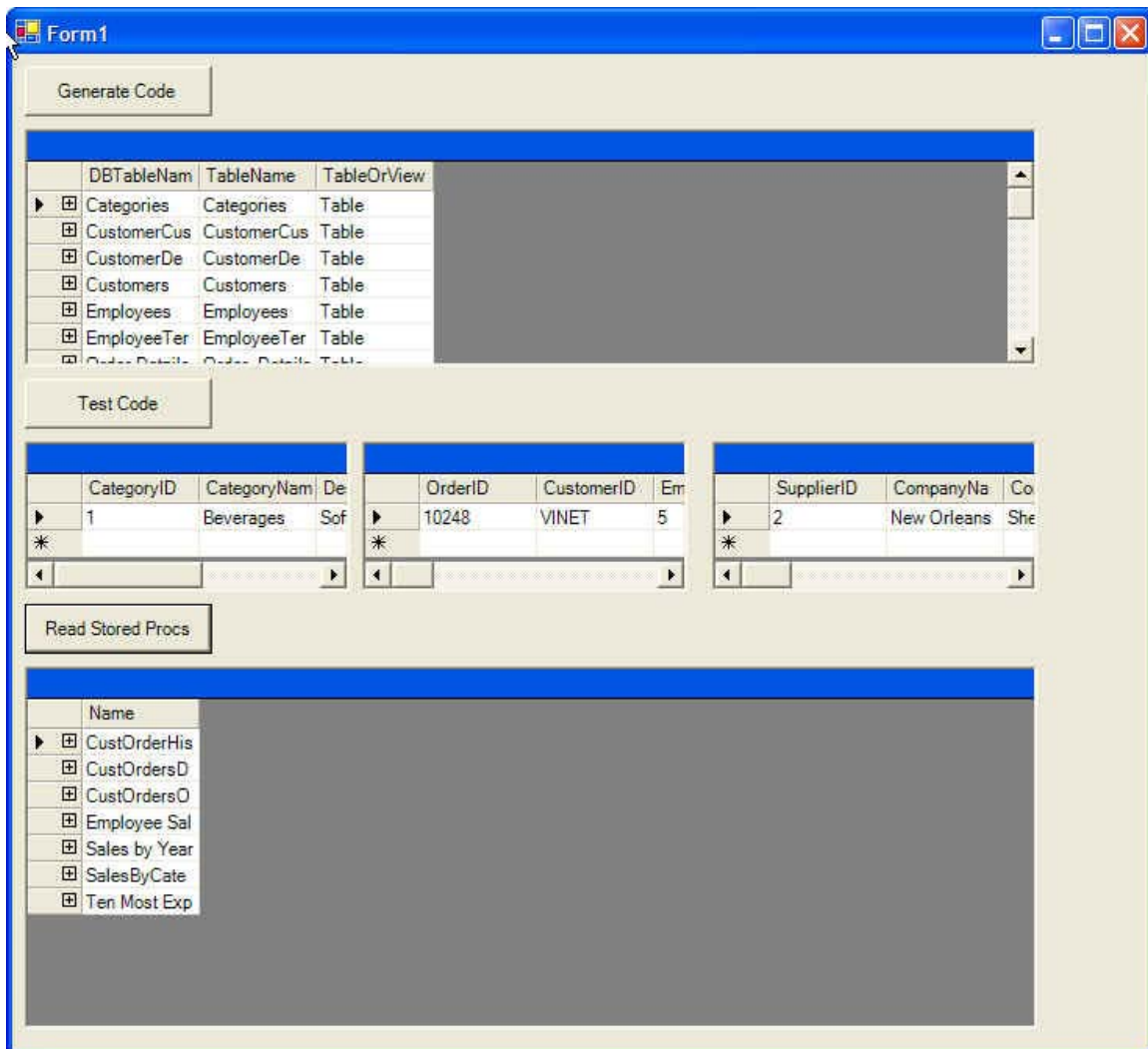
// Deserialize the TableCollection from the file and
// assign the reference to a local variable.
MonstarilloTypes.TableCollection tables =
(MonstarilloTypes.TableCollection) formatter.Deserialize(fs);
fs.Close();
```

You can now use the tables variable in your code.

MonstarilloDBSample project

The MonstarilloDBSample project is a sample project that uses the MonstarilloDBObject. The project will show you how to use the MonstarilloDBObject in a real world example.

The sample project was installed with Monstarillo. Note that the connection string used to connect to the Northwind database is stored in the project's App.config file. Below is a screen shot of the MonstarilloDBSample project.



The sample project uses a MonstarilloDBObject generated against the Northwind database.

Pressing the Generate Code button will instantiate a MonstarilloDBObject and use it to create stored procedures to select data from all of the tables selected in the Monstarillo code generation run that created the MonstarilloDBObject and the

.Net code to call the stored procedures. The stored procedures will be in a file called TSQL.sql and the code in a file called Code.cs

The sample project has a class called GeneratedCode that uses the code generated in Code.cs. Pressing the Test Code button calls three of the generated methods and fills three DataGrids.

Pressing the Read Stored Procs button instantiates another MonstarilloDBObject and reads all of the Stored Procedures from it.

Tutorials

Tutorial 1 Creating Your First Template.

In this tutorial you will learn how to create your first Monstarillo template. You will create a template that will generate insert stored procedures for each of the tables selected in your database.

It is recommended that you create a new folder for your custom templates. Your new folder must have a templates.xml and a CreateGUI.xml. Both of these files can be copied from the templates folder that Monstarillo uses. When copying the templates.xml file all of the data between the </xs:schema> and </TemplateSets> tags can be deleted.

The easiest way to create a Monstarillo template is to start off with your finished product. In this case your finished product will be a stored procedure that will insert a record into a table. Below is the code your new template will create. It is an insert stored procedure for the categories table in the Northwind database.

```
if exists (select * from dbo.sysobjects where id = object_id(N'yb_Categories_Insert') and
OBJECTPROPERTY(id, N'IsProcedure') = 1) drop procedure yb_Categories_Insert
```

```
go
CREATE PROCEDURE [dbo].[yb_Categories_Insert]
    @CategoryName nvarchar(30),
    @Description ntext,
    @Picture image
    ,@CategoryID int OUTPUT
    ,@ErrorCode int OUTPUT
AS
INSERT [dbo].[Categories]
(
    [CategoryName],
    [Description],
    [Picture]
)
VALUES
(
    @CategoryName,
    @Description,
    @Picture
)

Select @CategoryID = SCOPE_IDENTITY()
```

```
-- Get the error code
SELECT @ErrorCode=@@ERROR
GO
```

Notice that the transact sql code starts with a line that drops the stored procedure if it already exists. Our stored procedure is named in the following manner <prefix><table name><_Insert>.

Lets start with the first line of the template. We will replace the prefix with a %%SprocPrefix%% tag (Monstarillo will replace this tag with the stored procedure prefix entered into the GUI at runtime) and the table name with a %%TableName%% tag. See the tag reference for more information on Monstarillo tags.

```
if exists (select * from dbo.sysobjects where id =
object_id(N'%%SprocPrefix%%%%TableName%%_Insert') and
OBJECTPROPERTY(id, N'IsProcedure') = 1) drop procedure
%%SprocPrefix%%%%TableName%%_Insert
```

To generate CREATE PROCEDURE [dbo].[yb_Categories_Insert] you will replace the table name with the %%TableName%% tag.

Next we need to create a parameter for each of the updatable columns in the table. We need to add a parameter to return an error code and we need to make all of the identity columns output parameters. First we will loop through all of the updatable non identity columns. The following do so.

```
%%ForEachNonIdentityUpdatableColumnRemoveComma%%
    @%%ColumnName%% %%ColumnTSQLType%%,
%%EndFor%%
```

Notice that we are creating a parameter with the same name as the column preceded by an “@”. The %%ColumnTSQLType%% tag will give us the column’s transact sql type. Notice that the code in the loop is followed by a comma. The %%ForEachNonIdentityUpdatableColumnRemoveComma%% tag allows us to loop through all non identity updatable columns and removes the last comma from the generated code.

Next we will add an output parameter for each of the identity columns in the table and a parameter to return our error code.

```
%%ForEachIdentityColumn%%
    ,@%%ColumnName%% %%ColumnTSQLType%% OUTPUT
%%EndFor%%
,@ErrorCode int OUTPUT
```

Notice that each line in the loop starts with a comma.

Next we will add the code to generate the insert() portion of our stored procedure.

```
AS
INSERT [dbo].[%%DBTableName%%]
(
    %%ForEachNonIdentityUpdatableColumnRemoveComma%%
    [%%DBCColumnName%%],
    %%EndFor%%
)
```

Notice the %%DBTableName%% tag surrounded by brackets. The %%DBTableName%% uses the table name as used in the database which can contain spaces. Monstarillo will change the table name so that it is a legal .net identifier by replacing spaces with a under bar. You could access this new name with the %%TableName%% tag. The same is true for the %%DBCColumnName%% tag.

Next we will add the code to generate the values() portion of our stored procedure.

```
VALUES
(
    %%ForEachNonIdentityUpdatableColumnRemoveComma%%
    @%%ColumnName%%,
    %%EndFor%%
)
```

In this code we loop through the non identity updatable columns and print the name of each column's parameter.

Next we assign values to our identity columns and return our error code.

```
%%ForEachIdentityColumn%%
    Select @%%ColumnName%% = %%DBVersionIdentityReturn%%
%%EndFor%%

-- Get the error code
SELECT @ErrorCode=@@ERROR
GO
```

Notice the use of the %%DBVersionIdentityReturn%% tag. The %%DBVersionIdentityReturn%% tag returns SCOPE_IDENTITY() or @@Identity depending on the version of SQL server being used.

Your new template in its entirety is below:

```
if exists (select * from dbo.sysobjects where id =
object_id(N'%%SprocPrefix%%%%TableName%%_Insert') and
OBJECTPROPERTY(id, N'IsProcedure') = 1) drop procedure
%%SprocPrefix%%%%TableName%%_Insert

go
CREATE PROCEDURE [dbo].[%%SprocPrefix%%%%TableName%%_Insert]
    %%ForEachNonIdentityUpdatableColumnRemoveComma%%
    @%%ColumnName%% %%ColumnTSQLType%%,
    %%EndFor%%
    %%ForEachIdentityColumn%%
    ,@%%ColumnName%% %%ColumnTSQLType%% OUTPUT
    %%EndFor%%
    ,@ErrorCode int OUTPUT
AS
INSERT [dbo].[%%DBTableName%%]
(
    %%ForEachNonIdentityUpdatableColumnRemoveComma%%
    [%%DBColumnName%%],
    %%EndFor%%
)
VALUES
(
    %%ForEachNonIdentityUpdatableColumnRemoveComma%%
    @%%ColumnName%%,
    %%EndFor%%
)

    %%ForEachIdentityColumn%%
    Select @%%ColumnName%% = %%DBVersionIdentityReturn%%
    %%EndFor%%

-- Get the error code
SELECT @ErrorCode=@@ERROR
GO
```

Next we will modify the templates.xml file so that Monstarillo will recognize our new template. The templates.xml file is located in the templates directory under your Monstarillo home path. To have Monstarillo recognize a template set called Tutorial1 add the following to the templates.xml file.

```
<TemplateSet>
    <TemplateName>Tutorial1</TemplateName>
    <Database>SqlServer</Database>
    <Language>C#</Language>
    <NeedsGui>N</NeedsGui>
</TemplateSet>
```

To have Monstarillo recognize a template called test.mt add the following code. To your templates.xml file. The FileID is a unique number to define your template. FileName is the file name of your template. Your template must be placed in the same directory as the templates.xml file. The file extension is the file extension of your generated file. NewFileName is the name of your generated file. FolderName is the name of the folder your generated files will be added to. This folder will be created for you if it does not already exist. This folder will be created under the output directory selected in Monstarillo at runtime. If this value is left blank the files will be added to the output directory selected in Monstarillo at runtime. OverWriteExisting with a value of Y tells Monstarillo write over the generated file if it already exists. A value of N would prevent this. AccessModifier is a variable that you could use in your template. The %%AccessModifier%% tag returns the value of this column. The Append value of T tells Monstarillo to append the generated code to this file. Our template will generate a single file called InsertSprocs.sql in a folder called InsertStoredProcedures.

```
<TemplateFile>
    <FileID>1000</FileID>
    <FileName>InsertSproc.mt</FileName>
    <FileExtension>sql</FileExtension>
    <NewFileName>InsertSprocs</NewFileName>
    <FolderName>InsertStoredProcedures</FolderName>
    <OverWriteExisting>Y</OverWriteExisting>
    <AccessModifier></AccessModifier>
    <Append>T</Append>
</TemplateFile>
```

To add our template to our template set add the following to the templates.xml.

```
<TemplateSetFiles>
    <TemplateName>Tutorial1</TemplateName>
    <FileID>1000</FileID>
</TemplateSetFiles>
```

The TemplateSetName is the name of the template set and the FileID is the FileID of our template.

A CreateGUI.xml file is required in the template folder for code generation. Copy the CreateGUI.xml from the Monstarillo templates directory.

To have Monstarillo generate code from our template select the folder that contains your modified templates.xml file and run the Tutorial1 template.

Tutorial 2 Creating a Custom Tag

In this tutorial you will create your first custom Monstarillo tag. Once you create a custom tag you can use it just as you use the tags that ship with Monstarillo.

In this tutorial you will create a custom tag the prints out all of the properties for a column. Your custom tag will be a per column tag.

It is recommended that you create a new folder for when modifying or creating new templates. Your new folder must have a templates.xml and a CreateGUI.xml. Both of these files can be copied from the templates folder that Monstarillo uses. When copying the templates.xml file all of the data between the </xs:schema> and </TemplateSets> tags can be deleted.

Create a new Visual Studio Class Library project and call the project PerColumnTutorial.

Copy the MonstarilloTypes.dll from your Monstarillo directory to the debug directory of your new project. Add a reference to the MonstarilloTypes.dll.

Add a new class to your project called ColumnDictionary. Modify the class so it inherits from MonstarilloTypes.IMonstarilloPerColumnTag. Your new class should look like this:

```
public class ColumnDictionary:
MonstarilloTypes.IMonstarilloPerColumnTag
{
    public ColumnDictionary()
    {
        //
        // TODO: Add constructor logic here
        //
    }
}
```

Add a Name property to the new class.

```
public string Name
{
    get{return "PerColumnDictionary";}
}
```

Add a Method to print out each MonstarilloColumn property and its value. Notice the last line of the method which replaces %%ColumnDictionary%% with the code we generated.

```
public string PrintColumnDictionary(
MonstarilloTypes.IMonstarilloPerColumnContext context )
{
    // Create a String Builder to hold our code
    System.Text.StringBuilder sb = new System.Text.StringBuilder();
```

```

        // Append each property name and its value to the StringBuilder
        sb.Append( "ColumnName " + context.MonstarilloColumn.ColumnName +
Environment.NewLine);
        sb.Append( "ColumnDisplayName " +
context.MonstarilloColumn.ColumnDisplayName + Environment.NewLine);
        sb.Append( "DBColumnName " +
context.MonstarilloColumn.DBColumnName + Environment.NewLine);
        sb.Append( "ConvertToMethod " +
context.MonstarilloColumn.ConvertToMethod + Environment.NewLine);
        sb.Append( "IsForeignKey " +
context.MonstarilloColumn.IsForeignKey + Environment.NewLine);
        sb.Append( "IsIdentity " + context.MonstarilloColumn.IsIdentity +
Environment.NewLine);
        sb.Append( "IsPrimaryKey " +
context.MonstarilloColumn.IsPrimaryKey + Environment.NewLine);
        sb.Append( "IsUpdatable " + context.MonstarilloColumn.IsUpdatable
+ Environment.NewLine);
        sb.Append( "Length " + context.MonstarilloColumn.Length +
Environment.NewLine);
        sb.Append( "NumericPrecision " +
context.MonstarilloColumn.NumericPrecision + Environment.NewLine);
        sb.Append( "NumericScale " +
context.MonstarilloColumn.NumericScale + Environment.NewLine);
        sb.Append( "SQLClientType " +
context.MonstarilloColumn.SQLClientType + Environment.NewLine);
        sb.Append( "StoredProcedureVariableType " +
context.MonstarilloColumn.StoredProcedureVariableType +
Environment.NewLine);
        sb.Append( "TransactSqlType " +
context.MonstarilloColumn.TransactSqlType + Environment.NewLine);
        sb.Append( "ValueType " + context.MonstarilloColumn.ValueType +
Environment.NewLine);

        return context.Code.Replace( "%%ColumnDictionary%%" ,
sb.ToString() );
    }

```

Implement the PerformAction Method. Notice that our PerformAction method sets our context's code property to the output form a call to our PrintColumnDictionary method.

```

public void PerformAction(
MonstarilloTypes.IMonstarilloPerColumnContext context )
{
    context.Code = PrintColumnDictionary( context );
}

```

Build the new assembly and place it in the same folder as Monstarillo.exe. Modify the Plugins.xml by adding the following to it.

```

<PerColumnTag>
    <tag>%%ColumnDictionary%%</tag>
    <AssemblyName>PerColumnTutorial</AssemblyName>
    <ClassName>PerColumnTutorial.ColumnDictionary</ClassName>
</PerColumnTag>

```

Notice that the tag node is the value that we replaced in the PrintColumnDictionary method. The AssemblyName node is the name of our assembly and the ClassName is the name of our new class including the namespace.

Next we will create a new template that uses our new tag. Create a text file and add the following to it. Save it as test.mt in the Templates directory under the Monstarillo home directory.

```
%%ForEachColumn%%  
%%ColumnDictionary%%  
%%EndFor%%
```

Notice that we are looping through each column in the table and printing a column dictionary using our new custom tag.

Next we will modify the templates.xml file so that Monstarillo will recognize our new template. The templates.xml file is located in the templates directory under your Monstarillo home path. To have Monstarillo recognize a template set called Tutorial1 add the following to the templates.xml file.

```
<TemplateSet>  
    <TemplateSetName>CustomTag</TemplateSetName>  
    <Database>SqlServer</Database>  
    <Language>C#</Language>  
    <NeedsGui>N</NeedsGui>  
</TemplateSet>
```

To have Monstarillo recognize a template called test.mt add the following code. To your templates.xml file. The FileID is a unique number to define your template. FileName is the file name of your template. Your template must be placed in the same directory as the templates.xml file. The file extension is the file extension of your generated file. NewFileName is the name of your generated file. Monstarillo will replace "TableName" with the name of the table being processed at runtime. FolderName is the name of the folder your generated files will be added to. This folder will be created for you if it does not already exist. This folder will be created under the output directory selected in Monstarillo at runtime. If this value is left blank the files will be added to the output directory selected in Monstarillo at runtime. OverWriteExisting with a value of Y tells Monstarillo write over the generated file if it already exists. A value of N would prevent this. AccessModifier is a variable that you could use in your template. The %%AccessModifier%% tag returns the value of this column. The Append value of F tells Monstarillo to not to append the generated code to this file. Our template will generate a file for each table selected with the same name as the table with a txt extension in a folder called ColumnDictionary.

```

<TemplateFile>
    <FileID>1001</FileID>
    <FileName>test.mt</FileName>
    <FileExtension>txt</FileExtension>
    <NewFileName>TableName</NewFileName>
    <FolderName> ColumnDictionary </FolderName>
    <OverWriteExisting>Y</OverWriteExisting>
    <AccessModifier></AccessModifier>
    <Append>F</Append>
</TemplateFile>

```

To add our template to our template set add the following to the templates.xml.

```

<TemplateSetFiles>
    <TemplateSetName> CustomTag</TemplateSetName>
    <FileID>1001</FileID>
</TemplateSetFiles>

```

The TemplateSetName is the name of the template set and the FileID is the FileID of our template.

Run the template in Monstarillo. Your output will be similar to the text below.

```

ColumnName ShipperID
ColumnDisplayName ShipperID
DBCColumnName ShipperID
ConvertToMethod ToInt32
IsForeignKey False
IsIdentity True
IsPrimaryKey True
IsUpdatable True
Length 4
NumericPrecision 10
NumericScale 0
SQLClientType SqlInt32
StoredProcedureVariableType int
TransactSqlType Int
ValueType int

```

```

ColumnName CompanyName
ColumnDisplayName CompanyName
DBCColumnName CompanyName
ConvertToMethod ToString
IsForeignKey False
IsIdentity False
IsPrimaryKey False
IsUpdatable True
Length 80
NumericPrecision 0
NumericScale 0
SQLClientType SqlString

```

StoredProcedureVariableType nvarchar(80)
TransactSqlType NVarChar
ValueType string

ColumnName Phone
ColumnDisplayName Phone
DBCColumnName Phone
ConvertToMethod ToString
IsForeignKey False
IsIdentity False
IsPrimaryKey False
IsUpdatable True
Length 48
NumericPrecision 0
NumericScale 0
SQLClientType SqlString
StoredProcedureVariableType nvarchar(48)
TransactSqlType NVarChar
ValueType string

Tag Reference

ForEach Tags

ForEach tags are used to process a group of columns or parameter that have something in common. A user may want to have a block of code generated for each column in a table or for each primary column in a table. ForEach tags are always followed by a `%%EndFor%%`

An example of a ForEach tag being used in a template is :

```
%%ForEachColumn%%  
    data.%%ColumnName%% = dt.Rows[0][%%DBColumnName%%];  
%%EndFor%%
```

The Monstarillo looping tags are defined below.

ForEachColumn

This tag is to loop through each column in a table or view.

ForEachColumnRemoveComma

This tag is to loop through each column in a table or view. It differs from the ForEachColumn tag in that it strips off a trailing comma. This tag is used in parameter declarations in stored procedures.

ForEachNullableColumn

This tag is to loop through each nullable column in a table or view.

ForEachNonNullableColumn

This tag is to loop through each non nullable column in a table or view.

ForEachIdentityColumn

This tag is to loop through each identity column in a table.

ForEachIdentityColumnRemoveComma

This tag is to loop through each identity column in a table. It differs from the ForEachIdentityColumn tag in that it strips off a trailing comma. This tag is used in parameter declarations in stored procedures.

ForEachNonIdentityUpdatableColumnRemoveComma

This tag is to loop through each identity column of a given table that is updatable. Columns are made not updatable at runtime through the Monstarillo GUI.

ForEachNonIdentityUpdatableColumn

This tag is to loop through each updatable column that is not an identity column. Columns are made not updatable at runtime through the Monstarillo GUI.

ForEachNonIdentityColumn

This tag loops through each column of a given table that is not an identity column.

ForEachNonIdentityColumnRemoveComma

This tag loops through each column of a given table that is not an identity column. It differs from the ForEachNonIdentityColumn data in that it strips off a trailing comma. This tag is used in parameter declarations in stored procedures.

ForEachTextColumn

This tag loops through each text column of a given table.

ForEachImageColumn

This tag loops through each image column of a given table.

ForEachPrimaryIdentityColumn

This tag loops through each column in a given table that is both a part of the table's primary key and an identity column.

ForEachPrimaryNonIdentityColumn

This tag loops through each column in a given table that is both a part of the table's primary key column and not an identity column.

ForEachUpdatableColumn

This tag loops through each column in a given table that is updatable.

ForEachUpdatableColumnRemoveComma

This tag loops through each column in a given table that is updatable. It differs from the ForEachUpdatableColumn in that it strips off a trailing comma. This tag is used in parameter declarations in stored procedures.

ForEachUpdatableNonNullableNonIdentityColumn

This tag loops through each column in a given table that is not nullable and is not an identity column.

ForEachPrimaryColumn

This tag loops through columns that make up the table's primary key.

ForEachPrimaryColumnRemoveComma

This tag loops through columns that make up the table's primary key. It differs from the ForEachPrimaryColumn in that it strips off a trailing comma. This tag is used in parameter declarations in stored procedures.

ForEachNonPrimaryColumn

This tag loops through each column in a table that is not a part of the table's primary key.

ForEachNonPrimaryColumnRemoveComma

This tag loops through each column in a table that is not a part of the table's primary key. It differs from the ForEachNonPrimaryColumn tag in that it strips off a trailing comma. This tag is used in parameter declarations in stored procedures.

ForEachPrimaryNonIdentityColumnRemoveComma

This tag loops through each column in a table that is not a part of the table's primary key and is not an identity. This tag is used in parameter declarations in stored procedures.

ForEachForeignKeyPKOnly

Loops through each foreign key entry where the table being processed is the parent of the foreign key relationship. See foreign key tags for more info.

ForEachForeignKeyFKOnly

Loops through each foreign key entry where the table being processed is the child of the foreign key relationship. See foreign key tags for more info.

Column Tags

Column tags emit data that describe a column in a table or view.

ColumnName

This is the name of a given column. This may not be the name of the column in the database. Monstarillo will attempt to make a column name legal in code by getting rid of spaces etc.

DBColumnName

This is the name of a given column. This will be the name of the column in the database and should be used to reference columns in stored procedures or columns in DataTables.

ColumnSQLType

This is the column type as reported by Sql Server.

ColumnTSQLType

This is the column's Transact Sql type to be used in transact sql.

Length

This is the column length as reported by the database.

SqlCodeType

This is the column's to define a SqlDbType.

NetCodeType

This is the column's native code type in .Net.

SqlCastType

When this column is returned in a DataTable this is the SqlType that the corresponding DataColumn can be cast into. See the C#PersistBaseInfo.mt template for an example.

TableName

The name of the table that the column is in. This may not be the name of the table in the database. Monstarillo will attempt to make a column name legal in code by getting rid of spaces etc.

DBTableName

The name of the table that the column is in. This will be the name of the table in the database.

NumericPrecision

The column's numeric precision as reported by the database.

NumericScale

The column's numeric scale as reported by the database.

IsNullable

The column's IsNullable property as reported by the database.

ColumnDisplayName

The column's display name. By default this is the same as the column name. This property can be modified at runtime by the user in Monstarillo.

Stored Procedure Tags**SqlCodeType**

The column's to define a SqlDbType.

SqlCastType

When this column is returned in a DataTable this is the SqlType that the corresponding DataColumn can be cast into.

SprocParameterList

Returns a string that will become the parameters of a .Net method call to execute a stored procedure using @ColumnName as a parameter name.

SprocNameCode

Returns the name of the stored procedure in a format that can be used in code. Spaces have been removed from the stored procedure name and replaced with underbars if needed.

SprocName

The name of the stored procedure..

ParameterName

The name of the stored procedure.

ParameterCodeName

The name of the stored procedures. procedure in a format that can be used in code. Spaces have been removed from the stored procedure name and replaced with underbars if needed.

NumericScale

The numeric scale of the stored procedure parameter as reported by the database.

NumericPrecision

The numeric precision of the stored procedure parameter as reported by the database.

NetCodeType

This is the column's native code type in .Net.

Length

The length of the stored procedure parameter as reported by the database.

ColumnTSQLType

The Transact Sql type of the stored procedure parameter.

ColumnSQLType

This is the column type as reported by Sql Server.

Foreign Key Tags

Foreign key tags are used to retrieve foreign key information for a given table.

FK_PrimaryKeyTable

The name of the table on the primary key side of the relationship in a format that can be used in code. Spaces have been removed from the stored procedure name and replaced with underbars if needed.

FK_PrimaryKeyTableDb

The name of the table on the primary key side of the relationship.

FK_PrimaryKeyColumn

The name of the column on the primary key side of the relationship in a format that can be used in code. Spaces have been removed from the stored procedure name and replaced with underbars if needed.

FK_PrimaryKeyColumnDb

The name of the column on the primary key side of the relationship

FK_ForeignKeyTable

The name of the table on the foreign key side of the relationship in a format that can be used in code. Spaces have been removed from the stored procedure name and replaced with underbars if needed.

FK_ForeignKeyTableDb

The name of the table on the foreign key side of the relationship

FK_ForeignKeyColumn

The name of the column on the foreign key side of the relationship in a format that can be used in code. Spaces have been removed from the stored procedure name and replaced with underbars if needed.

FK_ForeignKeyColumnDb

The name of the column on the foreign key side of the relationship

FK_PrimaryKeyColumnType

The column type of the column on the primary key column side of the relationship.

FK_ForeignKeyColumnType

The column type of the column on the foreign key column side of the relationship.

FK_SqlBefore

Returns the string needed to put before the value of the column in question in a sql statement. If the column type is string a quote.

FK_SqlAfter

Returns the string needed to put after the value of the column in question in a sql statement. If the column type is string a quote

Global Tags**AccessModifier**

Returns the access modifier set in the Template.xml

AllColumnsList

Returns a list of all column names separated by a comma.

Example:

FirstName, LastName, ID

AllColumnsListIdentityOutput

Returns a list of all column names separated by a comma with identity column name prefixed with out in C# and ByRef in VB.

Example:

FirstName, LastName, Age, out ID

AllColumnsListWithNativeTypes

Returns a list of all column native types and column names separated by a comma.

Example:

string FirstName, string LastName, int ID

AllColumnsListWithNativeTypesIdentityOutput

Returns a list of all column native types and column names separated by a comma. Identity columns will be prefixed with out in C# and ByRef in VB.

Example:

string FirstName, string LastName, out int ID

AllColumnsListWithTypes

Returns a list of all column SqlTypes and column names separated by a comma.

Example:

SqlString FirstName, SqlString LastName, SqlInt ID

AllColumnsListWithTypesIdentityOutput

Returns a list of all column SqlTypes and column names separated by a comma.

Example:

SqlString FirstName, SqlString LastName, out ID

AllUpdatableColumnsList

Returns a list of all updatable column SqlTypes and column names separated by a comma.

Example:

SqlString FirstName, SqlString LastName, int ID

AllUpdatableColumnsListIdentityOutput

Returns a list of all updatable column SqlTypes and column names separated by a comma. Identity columns will be prefixed with out in C# and ByRef in VB.

Example:

SqlString FirstName, SqlString LastName, out ID

AllUpdatableColumnsListWithNativeTypes

Returns a list of all updatable column native and column names separated by a comma.

Example:

string FirstName, string LastName, int ID

AllUpdatableColumnsListWithNativeTypesIdentityOutput

Returns a list of all updatable column native and column names separated by a comma. Identity columns will be prefixed with out in C# and ByRef in VB.

Example:

string FirstName, string LastName, out int ID

AllUpdatableColumnsListWithTypes

Returns a list of all updatable column SqlTypes and column names separated by a comma.

Example:

SqlString FirstName, SqlString LastName, SqlInt ID

AllUpdatableColumnsListWithTypesIdentityOutput

Returns a list of all updatable column SqlTypes and column names separated by a comma.

Example:

SqlString FirstName, SqlString LastName, out SqlInt ID

ConnectionString

The connection string that Monstarillo used to query the database.

Database

The name of the database that Monstarillo used.

DBTableName

The name of the table currently being processed

DBVersionIdentityReturn

Returns SCOPE_IDENTITY() or @@Identity depending on the version of SQL server being used.

DeveloperName

Returns the developer name entered into the Monstarillo GUI at runtime.

FillDropDowns

Returns code to fill drop down lists for all columns with a control type of drop down list for the table currently being processed for templates using SqlTypes. Used in the code behind of an aspx page.

FillDropDownsNativeTypes

Returns code to fill drop down lists for all columns with a control type of drop down list for the table currently being processed for templates using Native types. Used in the code behind of an aspx page.

ForeignKeyIndexers

Returns code for Foreign key indexers in templates using SqlTypes and Info classes.

ForeignKeyIndexersMSAppBlock

Returns code for Foreign key indexers in templates using SqlTypes and MS data application block.

ForeignKeyIndexersNativeTypes

Returns code for Foreign key indexers in templates using native types.

IdentityColumnsListWithTypes

Returns a list of identity columns in the table being processed with SqlTypes separated by commas.

Example:

int ID

InsertParamColumnCount

Returns the number of updatable columns in the table currently being processed.

InsertParamColumnCountPlus1

Returns the number of updatable columns Plus 1 in the table currently being processed.

Namespace

Returns the base namespace as entered into the Monstarillo GUI at runtime.

NonIdentityColumnsListWithTypes

Returns a list of column name and their SqlTypes for columns that are not identity columns for the table currently being processed.

Example:

SqlString FirstName, SqlString LastName

NToNIndexersBusiness

Returns N:N Indexers business object code using SqlTypes for the current table in templates using info classes

NToNIndexersBusinessNativeTypes

Returns N:N Indexers business object code using native types for the current table in templates using info classes

NToNIndexersDABBusiness

Returns N:N Indexers business object code using SqlTypes for the current table in templates using Data Application Blocks.

NToNIndexersDABPersist

Returns N:N Indexers persist object code using SqlTypes for the current table in templates using Data Application Blocks.

NToNIndexersDABStoredProc

Returns N:N Indexers persist object code using SqlTypes for the current table in templates using Data Application Blocks.

NToIndexersInlinePersist

Returns N:N Indexers persist object code using native types for the current table in templates using Inline SQL.

NToIndexersPersist

Returns N:N Indexers persist object code using SqlTypes for the current table in templates using info classes.

NToIndexersPersistNativeTypes

Returns N:N Indexers persist object code using native types for the current table in templates using info classes.

NToIndexersStoredProc

Returns N:N Indexers persist object code using SqlTypes for the current table in templates using info classes.

PrimaryColumnCount

Returns the number of primary columns in the current table.

PrimaryColumnCountPlus1

Returns the number of primary columns plus one in the current table.

PrimaryKeyList

Returns a list of column names for columns in the current table's primary key.

Example:

ID

PrimaryKeyListForQueryString

Returns a string that can be used to create a query string to retrieve a record by it's primary key. See the WListCodeBehind.mt template for an example of it's usage

Example:

```
EmployeeID=<%# Server.UrlEncode( DataBinder.Eval(Container.DataItem, "EmployeeID" ).ToString() )
```

PrimaryKeyListWithSqlTypes

Returns a list of the current table's primary key columns with SqlTypes

Example:
int ID

SprocCodeName

The name of the stored procedure currently being processed in a format that can be used in code. Spaces have been removed from the stored procedure name and replaced with underbars if needed.

SprocName

The name of the stored procedure currently being processed.

SprocParameterList

Returns a list of the parameters of the stored procedure currently being processed.

SprocPrefix

Returns the stored procedure prefix as entered into the Monstarillo GUI at runtime.

TableName

The name of the table currently being processed in a format that can be used in code. Spaces have been removed from the stored procedure name and replaced with underbars if needed.

UpdatableParamColumnCount

Returns the number of updatable columns in the table currently being processed.

UpdatableParamColumnCountPlus1

Returns the number of updatable columns in the table currently being processed plus one.