DOCUMENT PURPOSE  
This document will provide developers with a common set of standards for developing DotNetNuke Modules. The organizational goal here is that developers use a similar style of programming to create a module so that when inspected by future developers will feel that it was created by a single developer. The benefits using this approach are both external and internal. The first being a case where a customer has purchased more than one module developed by Engage. The will see a consistent programming style and approach in all modules. The second is internally when other developers are brought on to help with a module they will use the same approach and style as every other module.

…

Contents

[DOCUMENT PURPOSE 1](#_Toc188257777)

[PROGRAMMING PRACTICES 1](#_Toc188257778)

[Naming Conventions 1](#_Toc188257779)

[Coding Styles 3](#_Toc188257780)

[SQL Server Development Styles 6](#_Toc188257781)

[STYLE PRACTICES 7](#_Toc188257782)

[Button or Link 7](#_Toc188257783)

[Using DotNetNuke Styles 7](#_Toc188257784)

# PROGRAMMING PRACTICES

## Naming Conventions

Below is a table of common naming conventions.

Please review the .NET class libraries for additional info, examples and further standards.

###### Variable Naming

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Casing | Example | Comments |
| Class | Proper | Policy |  |
| Enum | Proper | AddressType |  |
| Enum Values | Proper | AddressType.Work |  |
| Event | Proper | StatusChange |  |
| Exception | Proper | LoginException | Class name should end with the word ‘Exception’ |
| Class Level  Data members | Camel | \_parentTypeID |  |
| Interface | Proper | IAttributeContainer | Interface names always begin with “I” |
| Method | Proper | GetAttribute() |  |
| Method Arguments | Camel | GetAttribute(string attributeName) |  |
| Namespaces | Proper | Engage.Marketing |  |
| Property | Proper | ItemId | Same name as backing field, just without underscore and proper case |

**Note:** *Hungarian Notation was common at the time of Visual Basic 6.0 release. With the emergence of object oriented development and coding practices it has fallen out of favor. We will continue to use limited Hungarian Notation in Visual Basic 6.0 and Visual Basic .NET, but if this code is later replaced or reused in a new programming language or versions of existing programming languages it should be avoided.*

###### Control Naming

We still use Hungarian Notation for controls on ASP.NET controls and pages. Below are the prefixes to use here. Feel free to suppress any errors from code analysis about using Hungarian notation for controls and their event handlers.

|  |  |  |
| --- | --- | --- |
| Control Type | Prefix | Example |
| Textbox, DNN Text Editor | txt | txtName |
| Label, DNN Label | lbl | lblName |
| DropDownList | ddl | ddlUser |
| RadioButtonList | rbl | rblVisibility |
| CheckBoxList | cbl | cblVisibility |
| CheckBox | chk | chkVisible |
| RequiredFieldValidator | rfv | rfvName |
| CompareValidator | cv | cvSortOrder |
| RangeValidator | rv | rvSortOrder |
| RegularExpressionValidator | regex | regexPhoneNumber |
| CustomerValidator | val | valUser |
| Table, (HTML) table | tbl | tblUsers |
| TableRow, (HTML) tr | row | rowUser |
| TableCell, (HTML) td | td | tdUserId |
| Button, LinkButton, ImageButton | btn | btnSubmit |
| Image | img | imgUser |
| HyperLink | lnk | lnkProfile |
| ListBox | lst | lstUsers |
| HiddenField | hdn | hdnUserId |
| Literal | lit | litMessage |
| Calendar (ASP, AJAX, 3rd Party) | cal | calStartDate |
| MultiView | mv | mvPanes |
| View | vw | vwTopPane |
| PlaceHolder | ph | phControls |
| Panel | pnl | pnlUserInfo |
| UpdatePanel | upnl | upnlUserInfo |
| UpdateProgress | up | upSaveProgress |
| GridView | gv | gvTimesheet |
| DataGrid | dg | dgTimesheet |
| Repeater | rp | rpTimesheet |
| AJAX Control Toolkit Controls | ajax | ajaxRating |
| User Controls | uc | ucProfile |
| All others (SqlDataSource, etc.)  use first letter of each word | sds, etc. | sdsUserData |

## Coding Styles

All code should be consistent throughout a module. When reviewing an application’s source code all of the code should appear, barring differences in the names in the comments, to be written by the same developer.

###### Spacing

Place only a single statement on any line including Dim statements. Use four-space indentations to make your code readable. If working in a component that has already been written be sure to always follow the indentations and naming conventions already present in the component.

###### Comments

Use comments to explain code segments that are not obvious. Use the ‘\*\* …. \* comment delimiters for commenting your code as it will be useful for others.

###### Inheritance Structure

Make sure that each control in your module inherits from PortalModuleBase (in the DotNetNuke.Entities.Modules namespace) or a class that itself inherits from PortalModuleBase. For pages (.aspx), inherit from PageBase (in the DotNetNuke.Framework namespace).

###### Inline SQL Statements

All inline SQL statements should be made using a StringBuilder by appending the next part of the statement to the previous. Although this method is slower than continuing your existing string on the next line, it will make your code more readable and easier to debug line by line. Also, use full capitalization on SQL keywords and proper name casing on SQL Objects such as tables and views. Using StringBuilder objects will also help with the overall speed of the application as String objects are immutable and new String objects will be created with every “+” operation. Also, be sure to allocate the StringBuilder object large enough to hold the SQL statement. It’s preferable to make the StringBuilder larger than needed, because continuously reallocating the StringBuilder’s internal storage can be time consuming.

Use inline variables rather than directly putting values into the SQL statement. Also, use StringBuilder.AppendFormat to add the NamePrefix to object names from your module (NamePrefix is DatabaseOwner + ObjectQualifier + ModuleQualifier). Use DatabaseOwner and ObjectQualifier for DNN tables.

Example:

Dim sql as new StringBuilder(256)

sql.Append ("SELECT ObjectID, ShortDescription ");

sql.AppendFormat ("FROM {0}Object ", NamePrefix);

sql.Append ("WHERE ObjectID = @ObjectId");

###### Global Variables

Defining global or publicly scoped variables has in the past been standard practice. Since future system will or at least should use an object-oriented approach, avoid using public variables.

###### Enumerators

Enumerators should be used whenever possible to replace hard-coded type-id’s. This will make it easier to replace them if the database and/or lookup tables change.

Naming Conventions

All DotNetNuke scripts should be written using databaseOwner and objectQualfier tags. All database objects should also be prepended with a module qualifier, which is the name of the module, i.e. “EngageEmployment\_.”

###### Tables

Tables should be named to represent the data contained within them. They should be named using proper casing with new words delimited by uppercase of the first letter of the new word. Table names should not contain underscores and generally should not contain abbreviations or items specific to the system and unclear to a new developer. The one exception to this rule is lookup tables. All tables that are static and are lookups of certain types of data should be named as such and named with the prefix “lkp” followed by the proper casing of the full name of the type. Junction tables should include the names of both tables joined in full proper cased (non-abbreviated) names.

Examples:

ModuleName\_Object

ModuleName\_ObjectAttribute

ModuleName\_ObjectAffiliation

(Lookup Table) ModuleName\_lkpObjectType

(Junction Table) ModuleName\_ObjectRule

###### Columns

SQL Table columns should be descriptive to the data contained with the column. Column names should be in proper case. Identity columns should be specified as such and suffixed with “ID”. Identity columns should have the same name as the table followed by the ID extension. For example, the table Party will have an identity column named PartyID. The exception to this rule is lookup tables. Lookup tables’ identity columns should be named the same as the table without the lkp extension.

Examples:

ObjectID

AffiliationTypeID

ObjectTypeID (Lookup Table Identity Column)

###### Views

Views in SQL Server should be named with a description of what data the view represents and with a vw prefix. The vw prefix should be in lower case and the rest of the statement should be in proper case with words separated by capitalization. Words in view names should not be separated by an underscore.

Examples:

ModuleName\_vwParents

ModuleName\_vwParties

ModuleName\_vwCampaigns

ModuleName\_vwCampaignHierarchy

###### Stored Procedures

Stored procedures in SQL Server should be named with a description of what they are doing preceded by the lowercase sp prefix. They should not contain underscores because the SQL Server Engine uses underscores and SQL Server will first check the Master Database for these stored procedures. The names should be in proper case and should have a verb explaining what the stored procedure does followed by what data it is using or what it’s output is.

Examples:

ModuleName\_spAssignReferral

ModuleName\_spGetCompiledScript

ModuleName\_spAddParty

## SQL Server Development Styles

###### Data types

When developing an application you should use the correct SQL data type for the situation. For a complete list of SQL Data types see the Transact-SQL Overview Document. Follow these common styles for using data types in tables:

* Tables with a large amount of data should have identity columns of at least an integer data type.
* Lookup table column identities can generally be either a small int (if storing a lot of static values) or a tiny int.
* Bits should be used when needed if a Boolean situation is present. Bits are very fast and will improve performance.
* Bits data types should be avoided if there is a possibility of a trinary state.
* Use nvarchars data types instead of chars when needed to avoid extra length.
* Do not use text/ntext data types unless absolutely needed. Use the nvarchar datatype instead.
* Use the Unicode types for all situations by default (nvarchar instead of varchar, ntext instead of text).

###### Control of Flow

If multiple if… elseif… statements are needed in a stored procedure you should review the stored procedure as a whole. If code is being repeated often in the control flow sequence it may be time to re-factor and break each into separate procedure or break the code into dynamic inline SQL in Visual Basic.

###### String Manipulation

You should avoid using strings in SQL Stored Procedures that are concatenated to form executable statements. These do not compile in the stored procedure and since SQL handles strings inefficiently, should be written into dynamic inline SQL in Visual Basic.

# STYLE PRACTICES

## Button or Link

Use LinkButton or Hyperlink controls only for simple navigation, or where a browser button would be too large. Use Button controls whenever you are performing an action.

## Using DotNetNuke Styles

DotNetNuke includes a set of styles that are used throughout the DotNetNuke framework. In order to present a consistent user interface for Portal owners and to simplify style maintenance, modules must use the standard styles listed below for common UI elements. By using these styles, Skins can override the standard look of these predefined styles and know that the entire site will continue to provide a consistent user experience.

Style Class Description of use

|  |  |
| --- | --- |
| SubHead | Style of item titles on edit and admin pages |
| Normal | text style used for most text rendered by moduleslass Description of use |
| NormalDisabled | Text style used for rendered text which should appear disabled |
| NormalBold | Text style used for rendered text which requires emphasis |
| NormalRed | Text style used for error messages |
| NormalTextBox | Text style used for textboxes in the admin and edit pages |
| CommandButton | Text style for buttons and link buttons used in admin pages |