## Principles of Construction

There are four principal concerns in construction. They are, in order of precedence; security, quality, reuse and compliance with standards. This document will ensure that developers will produce a quality product consistently.

## General Guidelines

## Reusability

- Minimize code duplication.
- Be aware of and reuse existing controls and components wherever possible.
- Look for opportunity to create new reusable controls and components.
- Reusable code should be self-contained and should not have any external dependencies.
- Inherited object should be easy to expand and difficult to modify.
- Consider using Building blocks provided by Microsoft in Patterns and Practices.

## Application Security

- Data fields should always be declared as private so that they are not directly accessible out side the class.
- Declare get/set assessors to allow access to the data fields.

## Class / Method Functionality

- Each class should be responsible for carrying out only a small set of related tasks, making it easy to inherit from and difficult to modify.
- Each method should be responsible for carrying out only a single task.
- Use Overloading functions for backward compatibility wherever feasible.
- Avoid using Optional Parameters in VB.Net.
- Avoid writing very long methods. A method should typically have 1 to 25 lines of code. If a method has more than 25 lines of code, you must consider re factoring into separate methods.
- Method name should tell what it does. Do not use misleading names.

## Data Validation

- Data validation should be used to make sure data conforms to table schema before attempting to update the table.

## Presentation

- An application should have a consistent look and feel throughout.
- These guidelines may be decided on a case-by-case basis always keeping the customers needs in mind.
- Design consideration should include: MDI/SDI interface, Label size and font, Label and control orientation, etc

## Other Guidelines

- Do not hardcode numbers. Use constants instead.
- Do not hardcode strings. Use resource files.
- Avoid using many member variables. Declare local variables and pass it to methods instead of sharing a member variable between methods. If you share a member variable between methods, it will be difficult to track which method changed the value and when.
- Use enumerations wherever required. Do not use numbers or strings to indicate discrete values.
- Do not make the member variables public or protected. Keep them private and expose public/protected *Properties*.
- Never hardcode a path or drive name in code. Get the application path programmatically and use relative path.
- In the application start up, do some kind of "self check" and ensure all required files and dependencies are available in the expected locations. Check for database connection in start up, if required. Give a friendly message to the user in case of any problems.
- If the required configuration file is not found, application should be able to create one with default values.
- If a wrong value found in the configuration file, application should throw an error or give a message and also should tell the user what are the correct values.
- Error messages should help the user to solve the problem. Never give error messages like "Error in Application", "There is an error" etc. Instead give specific messages like "Failed to update database. Please make sure the login id and password are correct."

## Naming Conventions

A consistent naming pattern is one of the most important elements of predictability and discoverability in a managed class library. Widespread use and understanding of these naming guidelines should eliminate unclear code and make it easier for developers to understand shared code.

We will use mostly the following two approaches for our development purposes.

### Pascal Case Notation

The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized.

Examples: **BackColor, DataSet**

### Camel Case Notation

The first word is all lower case. For all other words in the name, the first letter is capitalized.

Examples: **lastName, firstName**

### Naming Guidelines

| Type | Case Notation | Example |
|---|---|---|
| Private Class Level Variable | Camel case prefixed with "_". | _firstName, _lastName |
| Protected Variable | Camel case | firstName, lastName |
| Public Variable | Pascal case | FirstName, LastName |
| Local Variable | Camel case | socialSecurityNumber, numberOfItems |
| Namespace | Pascal case or uppercase for very small word or abbreviations. | System.Windows.Forms, System.Web.UI |
| Class | Pascal case | Customer, Order, InvoiceDetail |
| Interface | Pascal case prefixed with the letter "I". | Examples: IServiceProvider, IFormatable |
| Parameter | Camel case | employeeName, numberOfItems |
| Method/Function | Pascal case | RemoveAll (), GetFormattedPhone() |
| Property/Enumerations | Pascal case | BackColor, NumberOfItems |
| Event | Pascal case notation suffixed with "EventHandler". | MouseEventHandler |
| Constant | Uppercase with words separated by underscores | AP_WIN_MAX_WIDTH, AP_WIN_MIN_WIDTH |
| Exception | "ex" | Catch (Exception ex) |
| Custom Exception | ClassnameException | PageNotFoundException |
| Project | *Projectname*Solution<br>*Projectname*DomainObjects<br>*Projectname*Web<br>*Projectname*Services | CTSSolution<br>CTSProject<br>CTSWeb<br>CTSServices |

## User Interface Control

Pascal case prefixed with the appropriate abbreviation as listed below.

| Control | Prefix | Control | Prefix |
|---|---|---|---|
| Button | btn | NumericUpDown | nud |
| CheckBox | chk | Panel | pnl |
| ComboBox | cbo | PictureBox | pbx |
| Container | ctr | ProgressBar | prg |
| ContextMenu | ctm | RadioButton | rdb |
| DataColumn | dcol | RichTextBox | rtf |
| DataGridDateTimePickerColumn | dgdtpc | Splitter | spl |
| DataGridTableStyle | dgts | SqlCommand | sqlcom |
| DataGridTextBoxColumn | dgtbc | SqlConnection | sqlcon |
| DataReader | dr | SqlDataAdapter | sqlda |
| DataRow | drow | StatusBar | stb |
| DataSet | dset | TabControl | tabctrl |
| DataTable | dtable | TabPage | tabpg |
| DateTimePicker | dtp | TextBox | txt |
| Dialog | dlg | ToolBar | tbr |
| DialogResults | dlgr | ToolBarButton | tbb |
| ErrorProvider | erp | ToolTip | ttp |
| GroupBox | grp | TrackBar | tkb |
| ImageList | iml | TreeView | tvw |
| Label | lbl | Timer | tmr |
| LinkLabel | lkl | UserControl | uc |
| ListBox | lst | XmlDocument | xd |
| ListView | lvw | MDI-Frame | frame |
| Mainmenu | mm | MDI-Sheet | sheet |
| MenuItem | mi | | |

In addition to providing naming standards,

- Method names should be a Verb or Verb Phrase
- Class/Property/Variable names should be Noun or noun phrase. In business objects, Boolean properties should have a Verb as the first part of the name such as IsActive, IsAvailable etc.
- Parameter names should have a descriptive name
- Attributes should always suffix with "Attribute"
- Exceptions should always suffix with "Exception"
- EventHandler should specify two parameters sender and e. Sender should always be object type, e should have the event class state

## Naming Guidelines for SQL

| Item | Case Notation | Example |
|---|---|---|
| Tables | Pascal case notation prefixed with a “Tab” and should be singular. | TabAddress |
| User Generated Stored Procedures | Pascal case prefixed with “Qry_” | Qry_GetAgencyKey |
| Report Stored Procedures | Pascal case prefixed with "Rpt_" | Rpt_Report1 |
| Views | Pascal case notation prefixed with a “Vw”. | VwPerson |
| Functions | Pascal case prefixed with “Fn_”. | Fn_ GetUserNameByKey |
| Triggers | Pascal case prefixed with "Tgr_" | Tgr_UpdateCounty |
| Input Parameters | Pascal case prefixed with “@i_”. | @i_Key |
| Output Parameters | Pascal case prefixed with “@o_”. | @o_Key |
| Variables | Pascal case prefixed with “@”. | @Key |

## Namespaces

A namespace is a way to categorize your classes and types so as to avoid naming conflicts between your code and other developer’s code. The recommended structure of a namespace is as follows:

Company.ProjectName[.Feature][.Design].

**Example:** **tct.ShoppingCart**

Feature and design are optional but useful in cases where you need to explicitly organize and separate functionality from the main project, particularly if the main project comprises multiple structures.

**Examples:** tct.ShoppingCart***.Accounting***

tct.ShoppingCart***.Shipping.WebUI***

tct.ShoppingCart***.Shipping.BusinessObjects***

## Code Formatting

### Indentation

- Comments should be in the same level as the code.
- Use TAB for indentation. Do not use SPACES.
- Use one blank line to separate logical groups of code.
- There should be one and only one single blank line between each method inside
- Use a single space before and after each operator and brackets.
- When an expression will not fit on a single line break it up according to these
    1) Break after comma.
    2) Break after operator.
    3) Align the new line with the beginning of the expression in the previous line.

Example:

**MethodCall(expr1, expr2, expr3, expr4, expr5);**

### Line Length

- Use common sense with line lengths.
- There are cases when it makes sense that a line should go past the visible IDE and force the use of scrollbars.
- However, these should be kept to a minimum and lines should be broken according to the guidelines above.

### Code Regions

Use the following code region names in the following order whenever possible. Region names should have a space prior to the first character and a space after the last character to increase readability. Also, when all regions are collapsed you should have one blank line between each region.

**Event Declarations**

Declare the event arguments classes and the events in this region.

**Private Class Instance Declarations**

Declare any external classes you are going to use

**Private Declarations**

Declare your private variables here. Remember to prefix your private class scope variables with an underscore ("_").

**Private Property Declarations**

Declare your private property declarations here. These should be prefixed with an underscore.

**Public Properties**

Place your public property get/set code here.

**Public Methods**

Place your public methods in this region.

**Private Methods**

Place your private methods in this region.

**Page Events**

Place any page events in this region.

**Individual Control's Events**

Add a new region for each control or group of controls below the Page Events region. For instance if you have a WebGrid on your page you will want a region named "WebGrid Events ."

Examples:

#Region " Constructors "

#End Region

#Region " Properties "

#End Region

#Region " Public variables "

#End Region

#Region " Private variables "

#End Region

#Region " Protected variables "

#End Region

#Region " Public methods "

#End Region

#Region " Private methods "

#End Region

#Region " Shared methods "

#End Region

**Web Page:**

#Region " Public variables "

#End Region

#Region " Private variables "

#End Region

#Region " Protected Variables "

#End Region

#Region " Properties "

#End Region

#Region " Events "

#End Region

#Region " Page Event Handlers "

#End Region

#Region " Control Event Handlers "

#Region " Buttons "

#End Region

#End Region

#Region " Validations "

#End Region

#Region " Public Methods "

#End Region

#Region " Private Methods "

#End Region

#Region " Protected Methods "

#End Region

**Web Service Class:**

#Region " Private Variables "

#End Region

#Region " Constructor "

#End Region

#Region " Private Methods "

#End Region

#Region " Protected Methods "

#End Region

#Region " Web Methods "

#End Region

# Code Commenting

## Inline Comments

- Use inline comments to explain assumptions, know issues, and algorithm insights.
- Do not use inline comments to explain obvious code. Well written code is self documenting.
- Comment each type, each non-public type member, and each region declaration.
- Use end-line comments only on variable declaration lines. End-line comments are comments that follow code on a single line.
- Do not create formatted blocks of asterisks that surround comments.
- Separate comments from comment delimiters (apostrophe) with one space.
- Begin the comment text with an uppercase letter.
- End the comment with a period.
- Include task-list keyword flags to enable comment filtering.
- Always apply C# comment blocks (///) to public, protected, and internal declarations.
- Use VB Comment Blocks(''') to all class, method, property, event declarations

## Class Header Description

The following comment blocks should appear at the top of each class module developed (before the class or interface keyword). Using this header will convey pertinent information to other developers.

```
/// <summary>
/// Base class used when needing a screen
/// that is divided into 4 sections.
/// Includes a top,bottom,left and right panel.
/// The left and right panels are
/// divided by a splitter bar control.
/// </summary>
/// <remarks>
/// <CodeWalkthroughHistory>
/// Reviewed By        Date Reviewed      Checklist Status
/// Sundar             10/10/2006         Passed
/// </CodeWalkthroughHistory>
/// <RevisionHistory>
/// Author             Date               Description
/// Viji               15/11/2006         Created Class
/// Shiek              20/11/2006         Added NodeSelected function
```

/// </RevisionHistory>

/// </remarks>

In the above example, replace /// with ‘ (tick) since you are coding in VB.Net

### Function Header Description

The following comment block should appear at the beginning of each method definition. Using this header will convey pertinent information to other developers.

/// <summary>

/// Function to update the text

/// displayed on the active node

/// </summary>

/// <param name="nodeToUpdate">Treenode that needs updating</param>

/// <returns>true if node was updated</returns>

/// <exception cref="System.Exception">Thrown when…</exception>

/// <RevisionHistory>

| | | |
|---|---|---|
| /// Author | Date | Description |
| /// Sundar | 10/10/2006 | Created function |
| /// Shiek | 10/10/2006 | Changed return type |

/// </RevisionHistory>

In the above example, replace /// with ‘ (tick) if you are coding in VB.Net

### User Message Guidelines

The purpose of this section is to setup guidelines for displaying effective and consistent message to the users through out the system. Displaying information using message box requires setting title, message, icon, buttons, and default button setting.

- MessageBox title should be descriptive of the action being performed, for example“Confirm Delete”, “File not found”. Never use “Information”, “Critical” and “Warning”
- MessageBox text should give the user a clear, non-technical explanation of the problem.
- MessageBox text should include steps or a clear explanation of how to prevent the problem from occurring again.
- MessageBox text should state the problem or goal first, then the solution.

| Message Type | Icons | Buttons | Default Button |
|---|---|---|---|
| Information | Information | OK | OK |
| Question | Information | Yes & No | No |
| System Error | Error | OK | OK |
| Other | Warning | OK | OK |

## Variables

### Declaration

- Declaring variable with proper scope. For example, do not declare variable globally if they are going to be used in one function.
- Declare variables at the top of the function, class, or sub.
- Use the simplest data type, list, or object required. For example, use Int over Long unless a 64 bit value will be stored.
- Declare member variables as private.
- Avoid specifying a type for an enum - use default of int unless you have an explicit need for long.
- Avoid using inline numeric literals (magic numbers). Instead, use a Constant or Enum.
- Avoid declaring inline string literals. Instead use Constants.
- Only declare constants for simple types such as int and string.
- Always explicitly initialize arrays of classes, interfaces, delegates, objects, and strings using for loop.

### Cleanup

- Destroy objects before exiting a function.
- Use Dispose method to destroy objects.
- Use the finally block to release resources.

## Error Handling

- Never declare an empty Catch block.
- Avoid nesting a try/catch within a catch block.
- Display the error message only once to the user. In case of exceptions, give a friendly message to the user, but log the actual error with all possible details about the error, including the time it occurred, method and class name etc.
- Use Constants for all application related error messages instead of hard coding.

# Code Walkthrough Guidelines

## Procedure

- A code walkthrough will be performed after the completion of each unit of work.
- The code walkthrough is a short process and should take no more than an hour or two for unit of work.
- The technical lead may review the code themselves or delegate this task to another team member.
- The code should be reviewed using the checklist
- Any related database items should be reviewed by the DBA/Technical Lead.
- Add name of reviewer, date reviewed, and current checklist status (passed, failed) to the top of the source code of the class/form/module.
- The developer should make any correction noted on the checklist.

## Source Code Documentation

A compact html (chm) file should be created for the code documentation. Every class along with its public methods, public properties and public events should have a brief documentation with examples.

## Source Code Checklist

| Items | Compiles | Comment |
|---|---|---|
| The source code complies with naming conventions | | |
| The source code is formatted correctly | | |
| Variables are declared using proper scope, type and initialization | | |
| Considering security, all attributes are declared with the proper scope identifier | | |
| Resources are released properly | | |
| The source code includes appropriate commenting | | |
| Reusable classes and components are used where possible | | |
| Error handling is used where appropriate | | |
| User messages comply with standards | | |
| GUI complies with the presentation standard of the project | | |
| All relevant database items comply with standards. | | |