OpenSpan Automation Coding Standards

By

Jeff Badger

Revision History

Date	Version	Section	Description	Author
March 2012	1.0	All	First Version	J Badger
April 2013	2.0	All	Revised to allow for multiple projects in solution and as a result of feedback from developers	A Tanaman
May 2013	2.1	14	Correction in cheat sheet	
September 2013	2.2	3	Clarification on global containers and naming	
September 2013	2.2	3	Clarification on private and public procedures	
September 2013	2.2	4	Allowing for exit from top-right port in specific conditions	
September 2013	2.2	14	Corrections in cheat sheet	
September 2013	2.2	15	Removed examples section, as empty	

Reviewers

Date	Sections	Name	Position/Department
Dec 2012-Feb 2013	All	Chris Mills, Jeff Badger, Peter Hall	

Distribution

Copy Number	Name	Location

Table of Contents

1 Introduction

With OpenSpan anybody can produce automations or even full applications. Initially, making working automations may seem easy. However, reliable and maintainable automations are more than simply working automations. Reliable and maintainable automations are easy to read, easy to debug and follow coding standards agreed upon by your organization. The following document is intended to be a starting point for creating your own OpenSpan coding standards. The naming conventions, coding standards and best practices in this document are derived OpenSpan's internal coding standards and various Microsoft and non-Microsoft guidelines.

2 Naming Conventions

- Always use descriptive names for global and local variables and parameters. Do not assume that the purpose of your variable or parameter is obvious.
- Automation names should tell what the automation does. If the automation name is obvious there is no need for additional documentation explaining what the automation does.
- Adapter names should be identical to their current application shortcut names. Since adapter names appear on the OpenSpan Runtime application menu and are visible to the user, these names should be meaningful.
- Use Pascal casing for component and automation names with no spaces or underscores (except where underscores form part of naming convention).
- Use Camel casing for variables and automation parameters with no spaces. For further guidance on capitalization, please refer to http://msdn.microsoft.com/en-us/library/ms229043.aspx.
- Prefix Boolean variables with "is" (isVisible, isNumeric).
- Use all caps for constants. Separate words with an underscore.
- Use appropriate prefix for UI elements to distinguish them from variables. Some common prefixes are listed below.
    - Prefix textboxes with "txt" (txtName, txtAddress)
    - Prefix comboboxes with "cmb" (cmbState, cmbSalutation)
    - Prefix checkboxes with "chk"(chkShipping)
    - Prefix radio buttons with "rad" (radGender)

3 Project Organization

- The preference is to have a separate project for each adapter so that they can be easily reused in other solutions. If implemented as separate projects, the UI will be in a StartUp project referencing the adapter projects.

- Each solution should have an automation at the root level of the StartUp project named Main. This automation should be respond to the RuntimeHostProjectStarted event and perform any required project initializations.

- Each solution should have a text file or Word document at the root level of the StartUp project named ReleaseNotes. This file will hold all documentation for each release.

- Each project in the solution should have a Global Container at the root level of the project:

    - In StartUp projects this should be called GlobalContainer
    - In Adapter projects this should be called *AdapterName*GlobalContainer

- All global variables and global components should be placed in this container. The following OpenSpan Advanced components should be added to this container initially – StringUtils and MessageDialog.

- The preference is to use custom components compiled as reusable assemblies rather than the Script component. However, if you use the Script component, you can have as many Script components as you like within each Global Container. Group scripts logically as if each group of script-methods were part of a class.

3.1 Adapter folders or projects

Depending upon whether you are creating multiple projects in the solution, create a folder or referenced project for each adapter.

3.1.1 Adapter folders

- Create a folder for each adapter and name the folder and the adapter the same. Under the adapter folder create the following subfolders – Events, Public and Private. You can also add a folder Services if applicable (see below).

- Event handler automations should be placed in the Events folder under the adapter which generates the event.

- Procedure automations should be placed in one of the following folders:

- The Public folder under the adapter that they reference if the procedure is called from another folder or project
  - The Private folder if it is intended to be called from within the current adapter folder

This distinction is primarily to highlight the public procedures, which control the work flow. Private procedures are intended to be used as support for public procedures.

- Web Services and Automation Broker Workers that call adapter procedures should be placed in the Services folder.

3.1.2 Adapter projects

- Create a project for each adapter and name the project as *Adapter*Project, where *Adapter* is the adapter's name.

- In the root create Events, Public, Private and Services folders as described above.

- Any global objects used by this project will reside in the project's own Global Container.

3.2 System folders

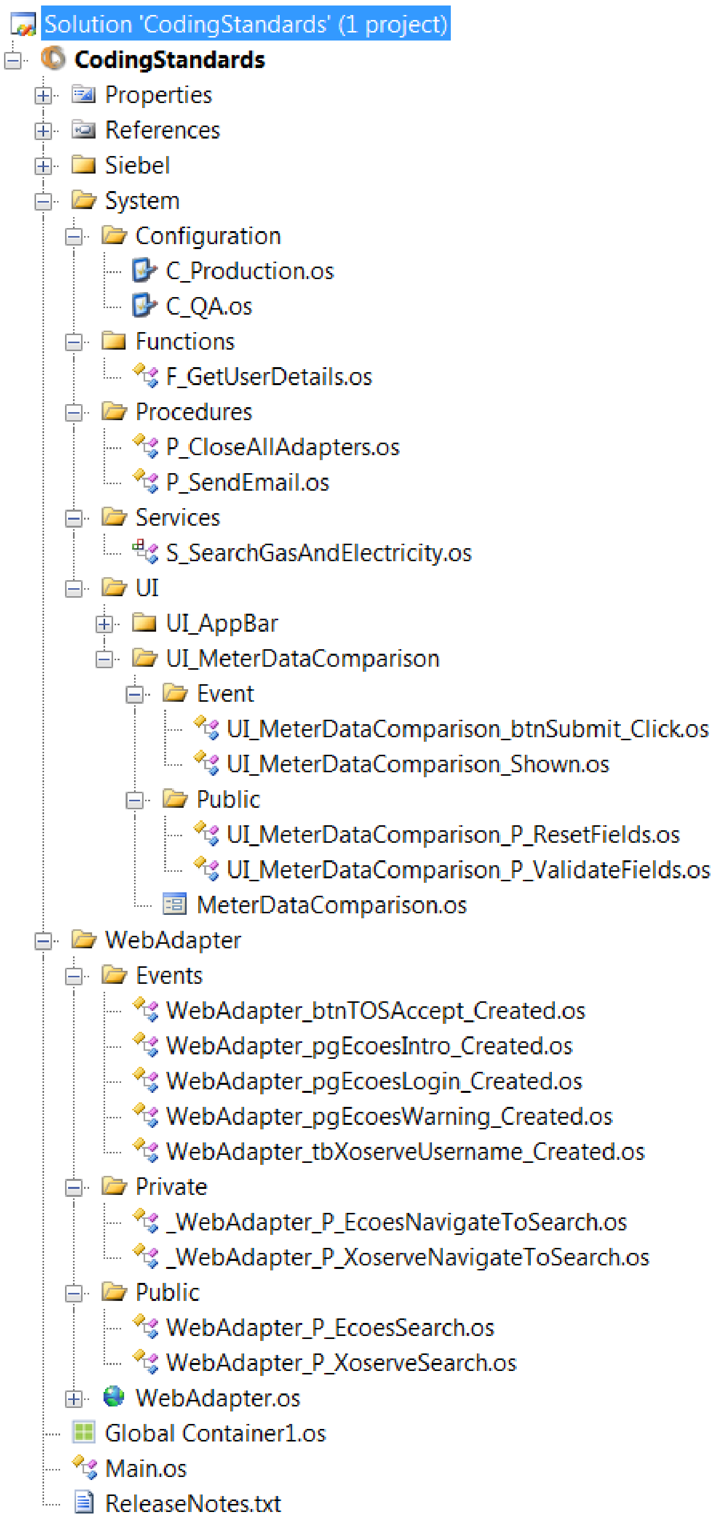
The project (or Startup project if the solution contains multiple projects) should have a System folder and a UI folder in the root:

- The System folder in the root with two subfolders – Configuration and Functions:
  - Configuration files should be placed in the System\Configuration folder.
  - Function call automations should be placed in the System\Functions folder. This folder may contain sub-folders to better organize these automations.

- Any project containing UI controls such as Window Forms or Application Bars should have a UI folder in the root:
  - Create a subfolder for each UI control and name the folder and the control the same. Under this, create the Events, Public, and Private subfolders.
  - The Events subfolder will contain event handler automations for the UI control.
  - The Public subfolder will contain system-wide procedures, which can call:
    - The UI control's private procedures
    - System functions
    - Adapter public procedures

- Any project containing Web Service and/or Automation Broker Worker automations should have a Services folder in the root:
  - Create a subfolder for each Service and name the folder and the Service the same. This folder will contain the Service. Under this, create the Public and Private subfolders.
  - The Public subfolder will contain the main automation, which can call:
    - The Service's private procedures
    - System functions
    - Adapter public procedures

This image shows the structure of a sample project:



```
Solution 'CodingStandards' (1 project)
CodingStandards
    Properties
    References
    Siebel
    System
        Configuration
            C_Production.os
            C_QA.os
        Functions
            F_GetUserDetails.os
        Procedures
            P_CloseAllAdapters.os
            P_SendEmail.os
        Services
            S_SearchGasAndElectricity.os
        UI
            UI_AppBar
            UI_MeterDataComparison
                Event
                    UI_MeterDataComparison_btnSubmit_Click.os
                    UI_MeterDataComparison_Shown.os
                Public
                    UI_MeterDataComparison_P_ResetFields.os
                    UI_MeterDataComparison_P_ValidateFields.os
                MeterDataComparison.os
    WebAdapter
        Events
            WebAdapter_btnTOSAccept_Created.os
            WebAdapter_pgEcoesIntro_Created.os
            WebAdapter_pgEcoesLogin_Created.os
            WebAdapter_pgEcoesWarning_Created.os
            WebAdapter_tbXoserveUsername_Created.os
        Private
            _WebAdapter_P_EcoesNavigateToSearch.os
            _WebAdapter_P_XoserveNavigateToSearch.os
        Public
            WebAdapter_P_EcoesSearch.os
            WebAdapter_P_XoserveSearch.os
        WebAdapter.os
    Global Container1.os
    Main.os
    ReleaseNotes.txt
```

4 Project Artifacts

4.1 Automations

Automations can be classified as event handlers, function/method calls or procedures.

- An event handler is triggered by a user or application action and may call a number of functions or procedures to perform the necessary work.

- Functions and procedures accept a set of input parameters and return a success or fail result and optionally may return values.

- A function is disconnected from the implementation and therefore does not reference any adapters or call any adapter procedures.

- A procedure may be used to perform functions for event handlers or to tie controls from more than one adapter together. There are several types of procedures:
    1. Global – references more than one adapter (typically associated with a UI control or a Web Service)
    2. Public – references one adapter and is intended to be called directly
    3. Private – references one adapter (or UI control) and is only called by Public procedures

4.1.1 Event handlers

Event handler automations should conform to the following:

- Each event should have a single event handler (i.e. a click event block should appear only once in a project). Normally, this means an automation for each event. In special cases where multiple events from a single adapter trigger a single action it is permissible to combine the events into a single automation.

- Event handler automations should be located in the Events folder under the adapter that generates the event.

- Event handler automations should be named using this naming convention –adapter prefix plus underscore plus control name plus underscore plus event (i.e. CRM_btn7_Click). In cases where multiple events from a single adapter are handled in a single automation, use this naming convention – adapter prefix plus underscore plus descriptive name for work being done (i.e. CRM_SetPatientNameVariable).

- Event handlers should call a procedure to perform work – no other actions should take place inside the event handler. Exceptions to this rule can be made when there are only a few steps to be taken and the code does not need to be unit tested.

4.1.2 Functions

Function call automations should conform to the following:

- Function calls names should include a prefix of F_.
- Function calls should be given descriptive names that clearly state what the function does.
- Function calls must have a single Entry Point and a single Exit Point.
- Function calls should perform only one job even if that job is very small.
- Function calls may only accept UI controls values only as parameters and should never directly access them.
- Function calls must always return a Boolean value indicating whether the automation completed successfully. Returning a value will allow for ease in creating unit tests for each function.
- Function calls should return any values that are computed by the function that may be needed elsewhere in the application with the Exit Point.
- Function calls should always have associated unit tests.

4.1.3 Procedures

Procedure automations should conform to the following:

- Procedures must have a single Entry Point and a single Exit Point.
- Procedure names should be given a prefix depending on the type of procedure:
  - Global – Optional prefix and underscore plus P_ (i.e. P_AddComments or SearchForm_P_PerformSearch)
  - Public – Adapter prefix plus underscore plus P_ (i.e. CRM_P_AddComments)
  - Private – Underscore plus adapter prefix plus P_ (i.e. _CRM_P_AddComments)
- Procedures must always return a Boolean value indicating whether the automation completed successfully. Returning a value will allow for ease in creating unit tests for each function.
- Procedures calls should return any values that may be needed elsewhere in the application with the Exit Point.
- Procedures may call other procedures and functions.
- Procedures may be used to tie controls from more than one adapter.

- Procedures may directly reference UI controls.
- Procedures should be given descriptive names that clearly state what the procedure does.
- Procedures should always have associated unit tests.

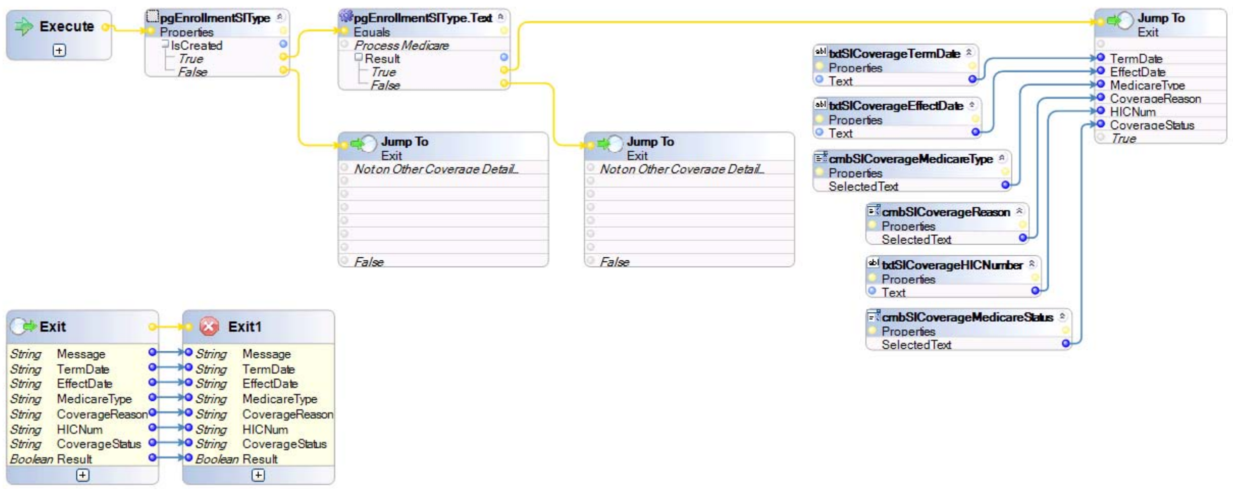
All automations should conform to the following:

- No more than one connector line should be run from a design block execution point.
- Automations should fit on a single screen when possible.
- All automation branches should be connected to something – a decision point should not be left unconnected. However, it is acceptable to do so if the execution flow has reached its end and control is passed to the Exit Point by using the top-right execution port of a component or automation (see next section)
- For readability – blue and yellow lines should not intersect whenever possible.

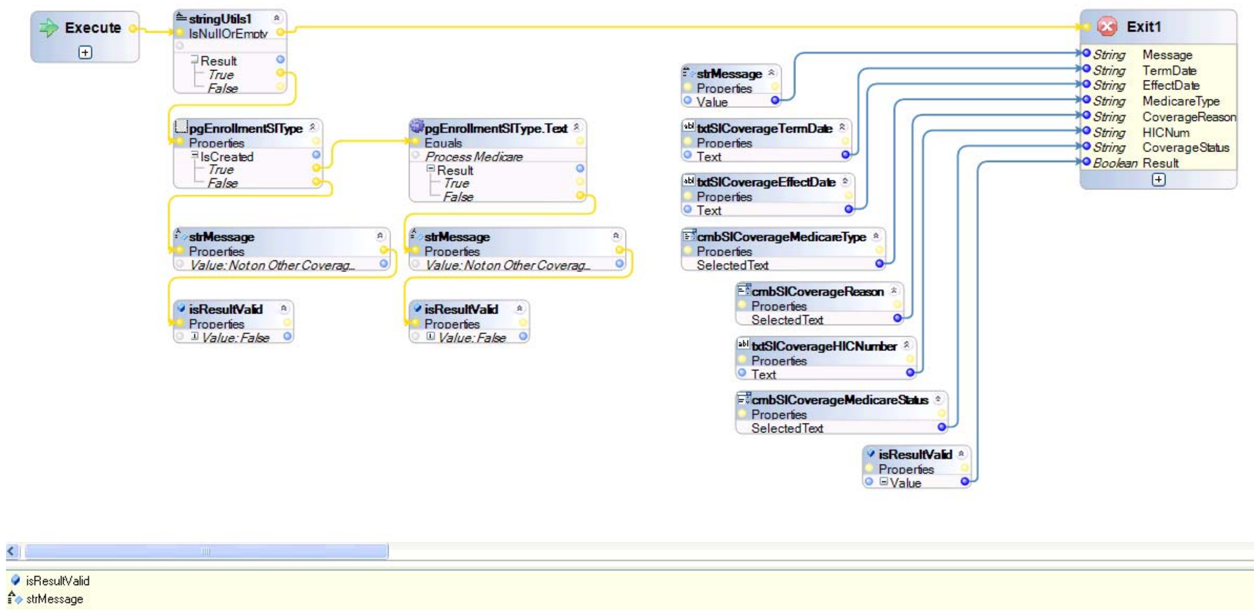
4.1.4 Exit Points

As mentioned, each procedure must have a single and guaranteed Exit Point. Control may be returned to the Exit Point by one of two methods:

- A jump label named Exit can be used to transfer control to the Exit Point and should be configured with the same parameters as the Exit Point:



- If the automation calls a component or another automation with an execution port on the top-right side, this can be used to transfer control to the Exit Point:



- The advantage of using labels is that it does not require using local variables to store values so that they can be passed to the Exit Point at the end of the automation. Additionally, the automation logic must be split off using a method (not a property): In the example above an 'IsNullOrEmpty" method with an always true result was artificially added for this purpose.

- On the other hand, labels require more system resources to handle and are less in-line with modern coding standards, as they simulate the "Goto" statement. It is also easier to miss a label call with the result that the automation will not complete via the Exit Point.

4.2 Configuration Files

If Configuration files are used in the project, they should conform to the following:

- Configuration files should include a prefix of C_.

4.3 Forms and Application Bars

If User Interface controls such as Forms and Application Bars are used in the project, they should conform to the following:

- User Interface controls should include a prefix of UI_.

- User Interface controls should be given descriptive names. In the case of forms, they should match the form title.

- Procedures referring to a specific User Interface should follow the same rules as detailed above in the Automations section. Specifically regarding naming:

    - Public – UI prefix plus underscore plus P_ (i.e. UI_MainForm_P_AddComments)
    - Private – Underscore plus UI prefix plus P_ (i.e. _UI_MainForm_P_AddComments)

4.4 Services

If Web Services or Automation Broker Workers are used in the project, they should conform to the following:

- Services files should include a prefix of S_.

5 Development Order

OpenSpan development should be done in a specific order:

- Develop functional requirements
    - Screenshots of application flow may be taken and notations made for trigger events and automation actions required.

- Interrogate all application controls required in the functional requirements
    - It is important to identify early in the development process if there are any required adapter controls which are not available.

- Design automations and test cases.

- Refactor automations as needed.

6 Interrogation

Interrogation should be performed prior to the development of automations. This step is used to verify that all of the functional requirements can be met. Any potential issues that are uncovered in the interrogation phase can be directed to OpenSpan Support and/or may possibly be deemed as out of scope for the current project.

Interrogation standards are as follows:

- Interrogate only the controls required.
- Rename controls as needed to meet naming conventions.
- Use Web Adapter Template to start web adapters (includes IE6 – IE8 web browser controls).
- Interrogate web applications using Global web pages unless otherwise required.
- Resolve any matching situations where interrogation did not produce a unique match.
- Review match rules for all controls on the object hierarchy
  - Review all Window Text rules.
  - Remove any rules which are not necessary.
- Check the object hierarchy after each screen transition to make sure that a control is not erroneously matching.
- Fully interrogate the application with more than one account or scenario.

7 Best Practices

- Watch for unexpected values – use Switch component default result.
- Convert strings to upper or lower case using StringUtils before doing comparison.
- Use StringUtils.IsNullOrEmpty to test instead of comparing to “”.
- Add a logging automation to perform diagnostic logging at key points in the application.

8 Error Handling

Error handling should be used in the following ways:

- Place Try-Catch around sections of automations that could fail – add descriptive error messages or handle the expected error automatically.
- Check validity of input data before performing processing (i.e. check zip code input for numeric).
- Use IsCreated or WaitForCreate before using controls which may not be matched.

9 Threading

Threading be should used carefully. Follow these guidelines:

- Do not use threading to improve performance.
- Avoid running automations on the user interface thread – make all execution paths off of the UI thread asynchronous.
- Use synchronous adapter events only when necessary and release them as soon as possible (i.e. “ing” events such as Clicking)
- Launch applications either on demand or in sequence rather than all at once.
- Do not use multiple threads when automating a single application – developers should attempt to simulate user behavior as most applications were designed for user interaction.
- No more than one connector line should be run from a design block execution point – use a parallel processing block to launch more than one thread from a design block.

10 Comments

ReleaseNotes document should contain release notes with release number, date of release and any pertinent documentation about the changes introduced in that release.

Each automation should have a comment block to document the automation functionality and contain a change log noting any changes made with the date and the name of the person making the change.

11 Testing

Unit testing should be done using OpenSpan's Unit Test facility, NUnit or another acceptable unit test platform. All function and procedure automations should have unit tests and each adapter should have unit tests for control matching.

12 Source Control

Using source control is recommended. OpenSpan does not support merging using source control – files should be locked while editing to prevent the need to manually merge versions.

13 Custom Components

Custom components should be used sparingly. All component source code should be in source control. The project's release notes should fully explain the use of the custom component and where the source code is located.

14 Appendix A: Cheat Sheet

Naming Conventions	Project Organization	Project Artifacts
CASING	*GlobalContainer*	SYSTEM ARTIFACTS
AutomationName	*Main*	**C_**ConfName
ComponentName	*ReleaseNotes*	**F_**FunctionName
automationParameter	Services\	**P_**ProcedureName
	S_Service1Name	**S_**ServiceName
variableAnyName	Private\	
isBooleanVariable	Public\	UI ARTIFACTS
THIS_IS_A_CONSTANT	*S_Service1Name*	**UI_**FormName_EventType
	S_Service2Name	**UI_**FormName_controlName_EventType
	System\	
UI ELEMENTS	Configuration\	**UI_**FormName_**P_**PublicProcedure
btnButton	Functions\	**_UI_**FormName_**P_**PrivateProcedure
divDiv	UI\	
chkCheckBox	UI_Form1Name\	
cmbComboBox	Events\	ADAPTER ARTIFACTS
frmForm	Private\	AdapterName_EventType
grpGroup	Public\	AdapterName_controlName_EventType
imgImage	*UI_Form1Name*	
pgPage	UI_Form2Name\	AdapterName_**P_**PublicProcedure
radRadioButton	Adapter1Name\	_AdapterName_**P_**PrivateProcedure
spnSpan	Events\	
tblTable	Private\	
tdTableCell	Public\	
trTableRow	*Adapter1Name***Adapter**	
tsTableSection	*Adapter1Name***GlobalContainer**	
txtTextBox		
winWindow	Adapter2Name\	