Sage 300 Web Screens SDK

Coding Patterns

August 2022

This is a publication of Sage Software, Inc.

Copyright © 2022. Sage Software, Inc. All rights reserved.

Sage, the Sage logos, and the Sage product and service names mentioned herein are registered trademarks or trademarks of Sage Software, Inc. or its affiliated entities. All other trademarks are the property of their respective owners.

Business Objects® and the Business Objects logo, BusinessObjects®, and Crystal Reports® are trademarks or registered trademarks of Business Objects Software Ltd. in the United States and in other countries. Business Objects is an SAP company.

Microsoft ®SQL Server®, Windows Vista® and the Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or in other countries. The names of all other products and services are property of their respective owners.

Your use of this document and the Sage product(s) described herein is governed by the terms and conditions of the Sage End User License Agreement ("EULA") or other agreement that is provided with or included in the Sage product. Nothing in this document supplements, modifies or amends those terms and conditions. Except as expressly stated in those terms and conditions, the information in this document is provided by Sage "AS IS" and Sage disclaims all express, implied or statutory warranties of any kind, including but not limited to the warranties of merchantability, fitness for a particular purpose or of noninfringement.

No implied license is granted to you under any Sage intellectual property or trade secret right. Sage reserves the right to revise, supplement or remove information in this document at any time without notice to you or others.

Sage End User License: [www.sage.com/us/legal/eula](http://www.sage.com/us/legal/eula)

Contents

[1. Overview 4](#_Toc524941624)

[2. C# Patterns 5](#_Toc524941625)

[2.1 Date Utility Pattern 5](#_Toc524941626)

[2.1.1 Examples 5](#_Toc524941627)

[2.2 Controller Unity Pattern 6](#_Toc524941628)

[2.2.1 An Incorrect Registration 6](#_Toc524941629)

[2.2.2 A Correct Registration 6](#_Toc524941630)

[3. JavaScript Patterns 7](#_Toc524941631)

[4. Razor View (cshtml) Patterns 8](#_Toc524941632)

1. Overview

Coding standards are a set of guidelines for a specific programming language that recommend programming style, practices and methods for each aspect of a program written in that language. Coding standards for the Sage 300c Web Screens are found in *All-In-One Code Framework Coding Standards* and the Sage 300 SDK Coding Standards documents of the Web SDK (<https://github.com/SageNADev/Sage300-SDK>).

Coding patterns describes a problem which occurs repeatedly and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice (*Christopher Alexander – father of the design pattern*).

This document will describe coding patterns for the Sage 300c Web Screens. The patterns will cover the C#, Razor View (cshtml) and JavaScript languages. This is a living document as new patterns will be created and existing patterns may continue to evolve.

The patterns are for use by the Sage 300 development team (internal, external and third party) for the purposes of:

* Reusability

The pattern must be reusable from multiple locations and thus encapsulation is important.

* Code and Defect Reduction

The pattern will reduce the size of the code base and potential defects by eliminating duplicate implementations.

* Efficiency

The pattern will allow developers to focus on the business logic.

The patterns also must focus on coding standards:

* Understandability

The pattern must be clearly readable, straightforward and well documented with both XML comments and comments for any code requiring additional clarification.

* Correctness

The pattern must behave as documented.

* Consistency

The pattern must follow all established coding standards.

* Modernization

The pattern must adopt current standards and practices found in the language.

1. C# Patterns
   1. Date Utility Pattern

The Date Utility Pattern is found in the DateUtil class of the Sage.CA.SBS.ERP.Sage300.Common.Utilities project.

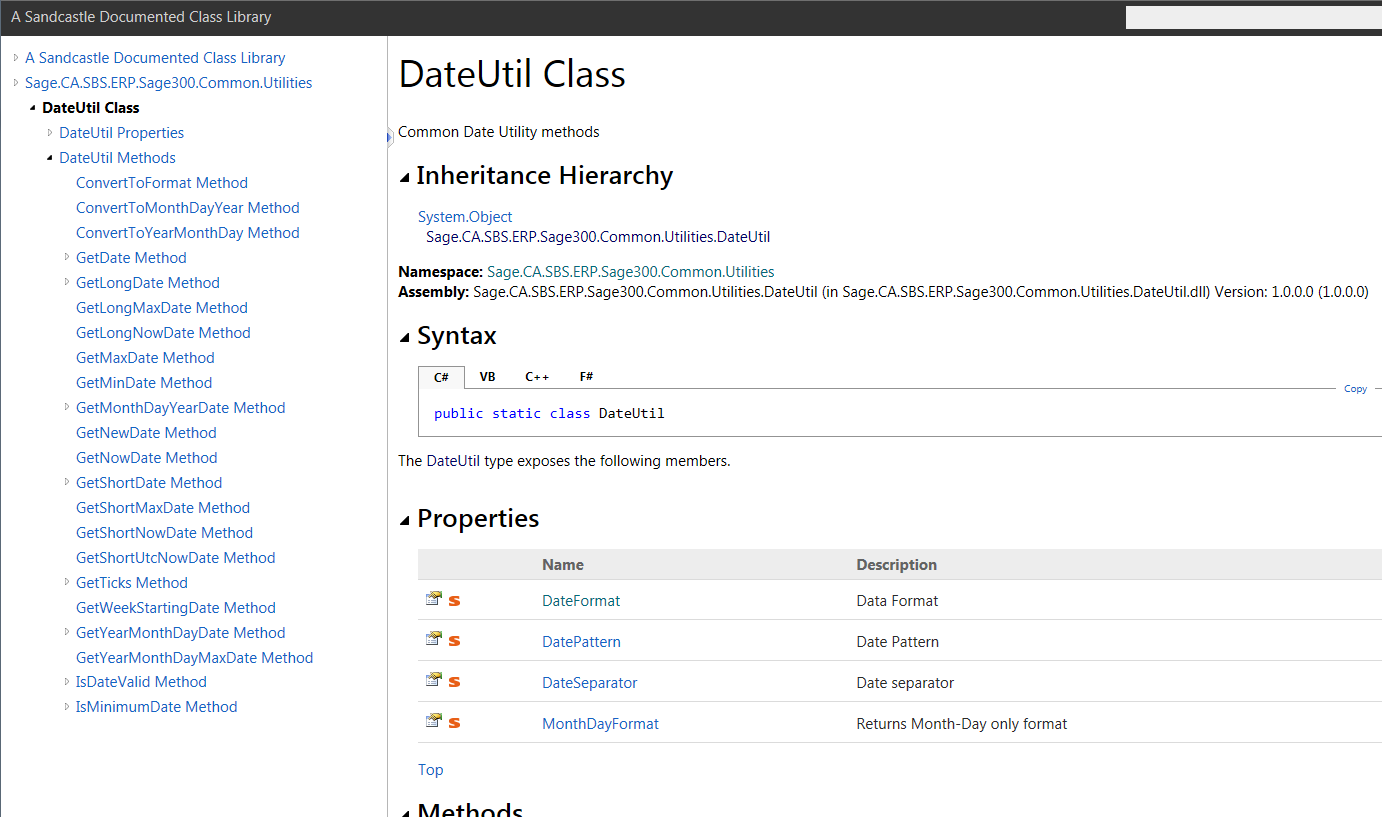
The purpose of this pattern is to encapsulate all functionality surrounding a date in C#.

Prior to this pattern, a developer was required to provide the locale to the DateTime object for proper localization of date formats. This was inefficient as every developer had to write this logic repeatedly. And, there were hundreds of duplicate routines as well.

The development and use of this pattern reduced over 6,600 lines of code from the code base.

The pattern gets dates, formats dates based upon locale, validates dates, minimum date logic is centralized, etc.

This pattern is to be used 100% of the time when dealing with dates. Do not create local date routines. Do not use intrinsic values and functions of the DateTime object. Do not format dates locally. Do not use CurrentLocale with dates. Do not write code that is already encapsulated in this class.



* + 1. Examples









* 1. Controller Unity Pattern

Services, Business Repositories and Controllers are registered in Unity for Dependency Injection (DI) purposes. But, identical named controllers in different modules can lead to collisions (i.e. AR Payment Codes and AP Payment Codes) if not registered correctly.

These collisions are prevented by prefixing the controller name added to the Unity container with the module id. The Controller Factory first checks Unity with the ***module + controller name*** and if it is not found it checks with just the ***controller name***.

* + 1. An Incorrect Registration

The following registration will have a collision if there is another ***TaxCodeMapping*** controller.

UnityUtil.RegisterType<IController, TaxCodeMappingController<TsRCode>>(container, "TaxCodeMapping");

* + 1. A Correct Registration

The following registration will not have a collision if there is another ***AccountSet*** controller. And, the Code Generation Wizard will automatically add the module prefix to the name being registered and therefore this is only an issue when the controller name is registered manually.

UnityUtil.RegisterType<IController, AccountSetController<AccountSet>>(container, "APAccountSet");

1. JavaScript Patterns
2. Razor View (cshtml) Patterns