Business Rules

NextAge Consulting
Pete Halsted
110 East Center St. #1035
Madison, SD 57042
pete@thenextage.com
www.thenextage.com
www.thenextage.com/wordpress

Table of Contents

BSD 3 License

NextAge Consulting - Pete Halsted

Pete Halsted has been developing custom business management applications for small to medium-sized companies, since 1987. His focus is on client/server, distributed and cloud based development utilizing WinDev, WebDev, and PostgreSQL. Pete is a Clarion Certified Developer with 25 years in the industry, has spoken at several Developers conferences, and provided Developer training and mentoring on a one on one basis. He has served companies both large and small as Project Manager, Lead Architect, Lead Developer and Chief Technology Officer. Pete tours the country full-time by motor home with his wife and dog, enjoying the freedom provided by cloud based technologies. Pete is available for Project Management, Custom Design, Development, Training, and Speaking assignments. For more information please visitwww.thenextage.com or follow his blog at www.thenextage.com/wordpress

General Information

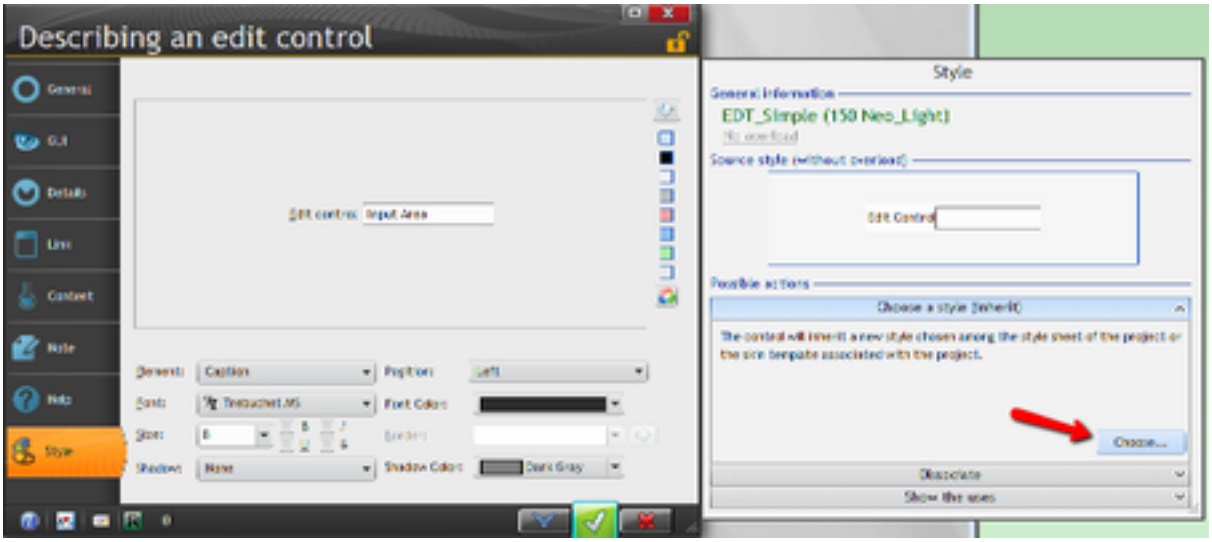
The Business Rule Class was created to provide a uniform method of enforcing business rules and data validation in an application. Furthermore it provides the ability to adjust and extend the business rules from the end user interface without recompiling the application. Allowing either a “super user” of the application to control the business rules, or the developer to ship different business rules tables, to different clients that make the same executable behave differently for each client. Default Rules are setup at design time, and written to the database when the application is ran, however these default rules can then be edited to change their behavior. Clearing the Rules table would have the effect of resetting all the rules to their default configuration. Validation errors are displayed using Balloon Tips, therefore the NextAge Balloon Tip class is required.

Setting up the Style Sheet

The class uses the change style command to switch the style of edit controls so that you can use a special style for required fields. Setting up this extra style is very easy. Once you create your project and have selected a skin do the following to create the two styles needed. The below steps will show you how to create a style similar to the one I use, which will show a Red border around edit controls that are required when they are not selected. You can of course get as creative as you want in configuring the Required Style versus your Normal Style.

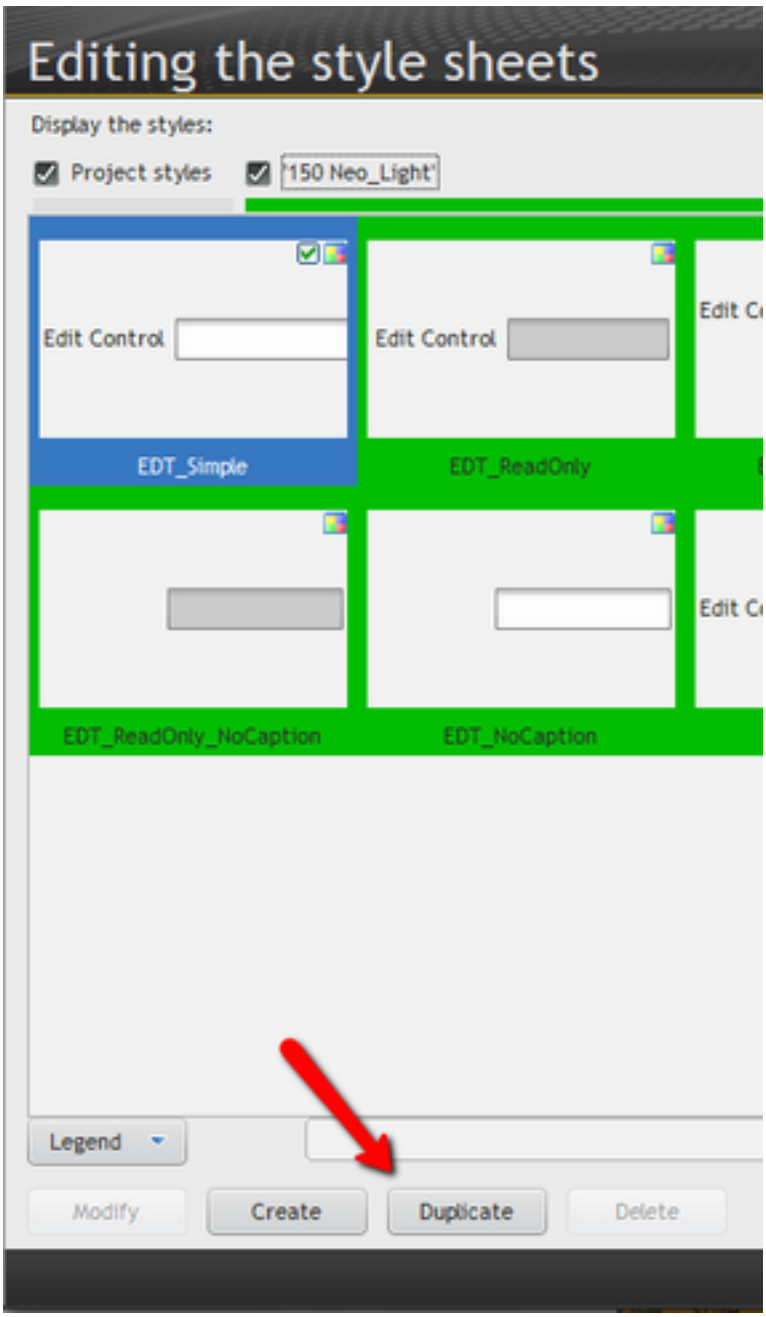
Note: If you are using the SCM you may have to check out your project style sheet prior to doing the following.

1. Edit the properties of an edit control that is using the “normal” style you will be using in your project.
2. Chose the Style tab.
    a. This will display the style properties and to the right a box that lets you chose a different style, or disassociate the control from the style.
3. Press the Choose Button



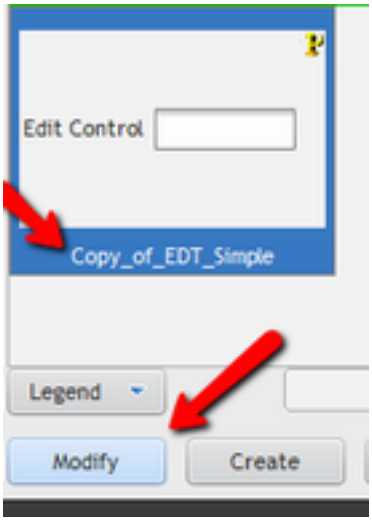
    a. This will show you a list of “Project Styles” and “Skin Styles”

4. Press the Duplicate Button

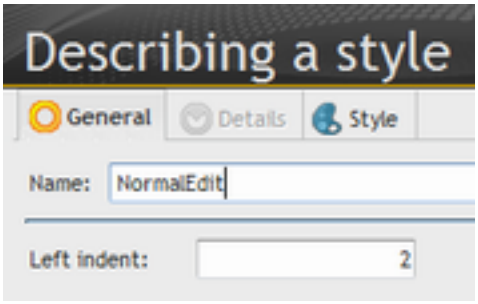


   a. This will create a new Project style identical to the current style of the control

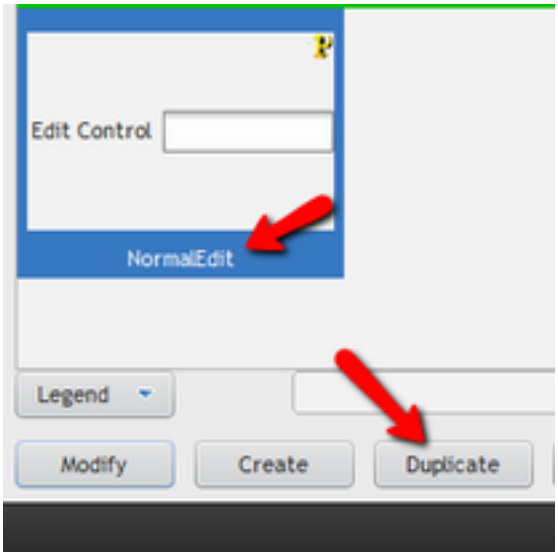
5. Select the Style just created and press the modify button



6. On the general tab rename the style to something that makes sense to you, such as "NormalEdit"

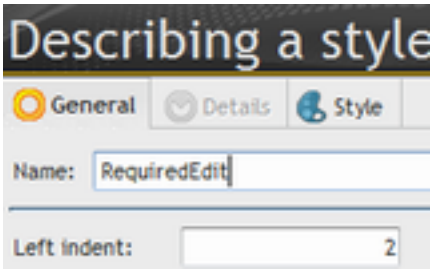


7. Press the Green Checkmark to save the changes to the style.
8. Press the Duplicate Button again to make another copy of the style.

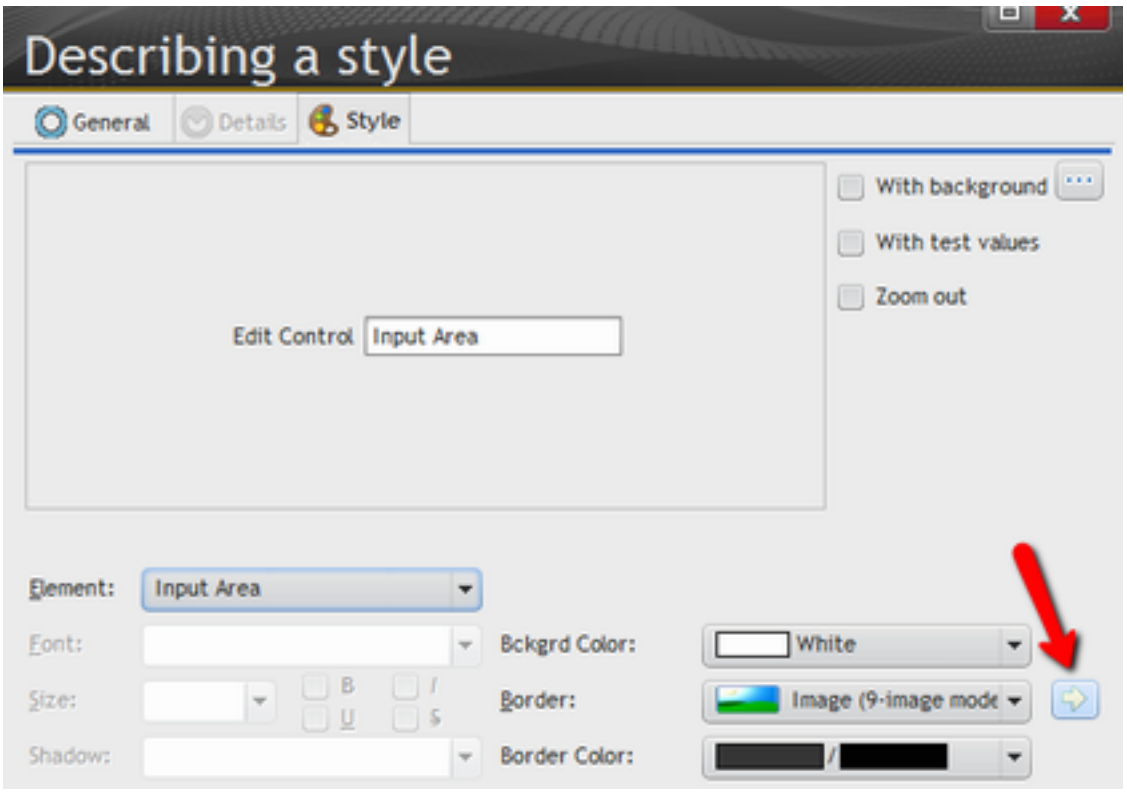


   a. You will be modifying this style to be used for required edit fields.
9. Select the style just created and press the modify button

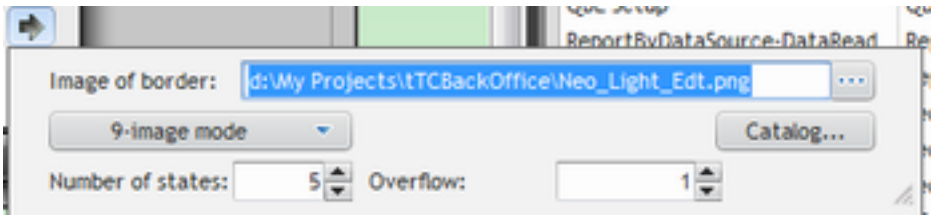
10. On the general tab rename the style to something that makes sense to you, such as “RequiredEdit”



11. Change to the Style tab
12. Select Input area from the Element drop down
13. The border option should say “Image (9 image mode)”
14. Press the Right Arrow button next to border option



    a. This will tell you the name of the image file being used to display the borders of the edit box.

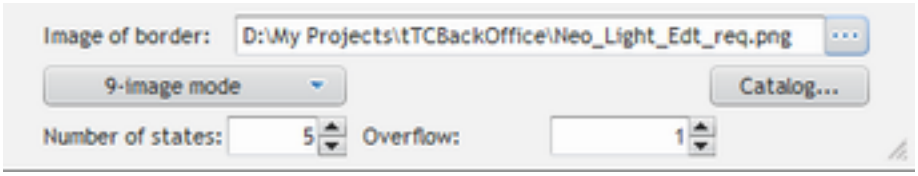


15. Make a copy of the image file from step 14

16. Edit the copy in your favorite image editor and change the first box to be a red border.

This will cause Required fields to show with a Red border on the screen when they are not selected.

17. Change the image file name in step 14, to the file you created in step 15 and 16.

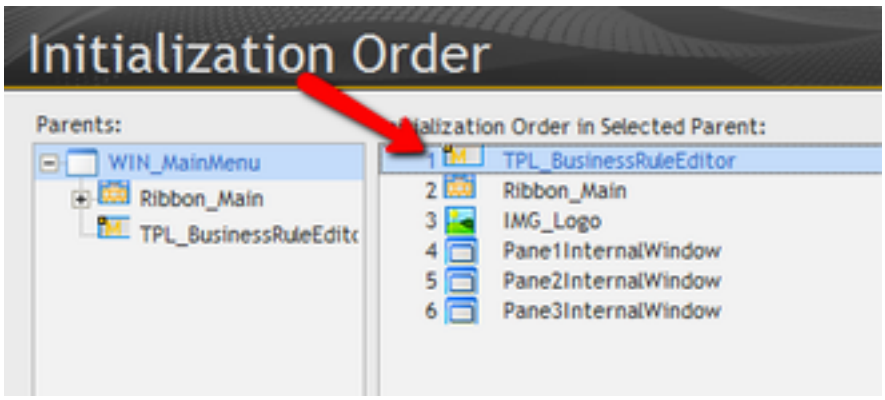


18. Press the Green Checkmark to save the changes to the style.

Setting up Business Rules for a Window

The recommend method for using the Business Rules class is to setup a Window Template to base all of your project windows on. This allows you to included the Business Rule Editor Button and a SetupDefaultRules procedure on every window. If not you will need to manually add them to the window.

1. Make sure the Business Rule Editor Button Template is the first item initialized by using the *Windows->Edit the Initialization Order* function.



2. If working with an Internal Window be sure to see the Note concerning Internal Windows and make the proper adjustments.
3. In the *Initialization Event* of the Window, call the ApplyBusinessRules method, this will apply any formatting or field level security setting for any rules on this Window. The *BRApply* Code Brick contains this code.
    a. `BR.ApplyBusinessRules(TPL_BusinessRuleEditor.WindowNumberForBr)`
        i. The control template contains the WindowNumberForBr variable, that was initialized when the button was initialized (this is why the button must be initialized first). This variable was returned by the Class call in the template and is the internal reference that the class uses to manage the fields and rules for this window.
4. For every control that will be managed by the Business Rules Class, a few lines of code need to be added, there are code bricks for all, and the code is very generic and should not need to be edited.
    a. *Initialization Event*
        i. `BR.AddControl(MySelf..FullName,TPL_BusinessRuleEditor.WindowNumberForBr)`
            1. This method is what adds the control to the class and allows it to be managed. As long as the Business Rule Editor Button template is consistently named it shouldn't need to be changed and can be populated with the *BRAdd* Code Brick
    b. *Exit Event*

i. *IF NOT BR.ValidateField(MySelf..FullName,False) THEN*
ReturnToCapture(MySelf)
END

1. This code actually performs the validation of the business rule. Again this code is generic and shouldn't need to be changed. It can be populated with the *BRValidate Code Brick*
2. When adding a button to the Business Rules for use with Field Level Security, there is not exit event since there will be not validation performed.

c. *Whenever Modified Event*
   i. For certain controls, such as Checkboxes, the validation should take place whenever the control is modified. For these fields place the Validate code in the *Whenever Modified Event* instead of the *Exit Event*. The code is identical in either case.

5. Prior to the Saving the record, Validate all fields with the following code:

```
BadField is string =
BR.ValidateAllFields(CTPL_BusinessRuleEditor.WindowNumberForBr)
IF BadField <> "" THEN
   SetFocus({BadField,indControl})
   RESULT False
END
RESULT True
```

This code forces the validation of all the controls for the window, if any of them fail focus is returned to that control. The *BRValidateAll* code brick contains this code. The supplied code returns a True or False, which is the expected result of the BeforeUpdateProcedure of the Browse Form Manager Class. Depending on where this code is placed and the programming style of the application it may need to be changed slightly.

6. Setup any Default Rules Desired
   a. Default Rules are exactly what they sound like, they are the default rule for a control. They are used to initially populate the Business Rule database.
   b. There are a few different forms of Default Rules that can be set, see the *Setting Default Rules section* for details

Note: Once a record exists in the Business Rule table it is used to regardless of any changes you make to the code. If you change the code and want the new default applied you must delete the record from the table so that it will be created with the new code. This behaviour is what allows Business Rules to be set or overridden at runtime.

Note: It is possible to add table columns to the Business Rule class by placing the same code in the events of the column. The Required Style will not be applied, but

field level security to Hide or Disable the control as well as any validation code, if the table is editable, will be executed.

Setting Default Rules

Default Rules are exactly what they sound like, they are the default rule for a control. They are used to initially populate the Business Rule database. Once a record exists in the Business Rule table it is used to regardless of any changes you make to the code. If you change the code and want the new default applied you must delete the record from the table so that it will be created with the new code. This behaviour is what allows Business Rules to be set or overridden at runtime.

There are a few different styles of Default Rules and code bricks exists for each to assist with the syntax of the calls. They all call the same method, DefaultRule, its just a matter of which parameters are passed or not.

1. **BRDefaultReq - Business Rules Setup Default Required**
    a. This sets up a field as being required and displays the required style (Red Box) around the field
2. **BRDefaultVal - Business Rules Setup Default With Validation**
    a. Includes custom validation code. See *Validation Code*
3. **BRDefaultSec - Business Rules Setup Default Field Level Security**
    a. Setups up Field Level security for a control. This of course only works if you are also using the Security Class.
4. The Possible Parameters to the DefaultRule Method are:
    a. ControlName
        i. The name of the control to place this Rule on.
    b. BrushColor
        i. Changes to background color of the control
    c. FontColor
        i. Changes to text color of the control
    d. Required
        i. Boolean True makes the control required.
    e. RequiredStyle
        i. Boolean True will use the Required Style when the control is required. See *Setting Up the Style Sheet*.
    f. ValidationCode
        i. Code to be executed for Validation. See *Validation Code*
    g. Category
        i. The Security Category Used for field level security
    h. AccessLevel
        i. The Access Level required for field level Security
    i. HiddenorDisabled
        i. If the security check fails the control is either eHidden (1) or eGrayed (2)

Note: Validation and required code uses “Soft Validation”, meaning if the value in the control is modified it is checked. This means the validation code is not executed if the user just tabs through the field. When the ValidateAllFields method is called it does “Forced Validation” and executes the validation code of all fields. This style of validation allows the user to navigate as they desire through the screen as long as when they save the record all fields pass validation.

Validation Code

Nearly any WX code can be place in the Validation Code, including referencing local variables and procedures of window and global variables and procedures of the application. This code can be as elaborate as desired. The only requirement is that if the validation passes the code must return an empty string. If the code returns a string, that string will be displayed as a balloon tip error on the field that failed validation.

Note: It may be helpful to test the validation code as standard WX code prior to moving it to a default business rule.

Special Note for Internal Windows

Since Internal Windows are not true windows but become part of the window they are used in, functions like mywindow return the name of the parent window, not of the Internal Window that you are currently working in. For this reason there is a little bit of special code that much be done for Internal Windows. In the Initialization Event for the Business Rule Editor Button you must manually set the name of the window before the code executes:

```
WindowNameForBR ="InternalWindowName"
ExecuteAncestor
```

By Placing the code before the Execute Ancestor it executes before the initialization code from the control template.

Other Requirements

This class uses the Balloon Tip Class to display the popup validation errors. The Balloon Tip class is another Open Source Class available from NextAge.

Database Table

The rules are stored in a database table. Therefore you must have a table configured for this purpose. The class has been written in a generic fashion that allows the file name and field names to be set via properties if desired. The class also uses Hxxxx commands for all file access, meaning that the class should work with any database that WX supports. The suggested definition for this file follows. If you do not use this definition you will have to use the associated properties to point to the correct values (*See the Special Properties*).

BusinessRules

- BusinessRulesID
  - Automatic ID
- FieldName
  - Text (100)
- BrushColor
  - Integer
- FontColor
  - Integer
- Required
  - Boolean
- RequiredStyle
  - Boolean
- Category
  - Text (20)
- AccessLevel
  - Integer
- State
  - Integer
- ValidationCode
  - Text Memo

Project Code

The class should be initialized in your project initialization code. There is a code brick to assist with this (BRSetup)

1. `BR is BusinessRules`
    a. Instantiates an instance of the class global for the entire project.
2. `BR.StyleSheet = "project.wdy"`
    a. The Style Sheet used by the project. This is the style sheet discussed above.
3. `BR.EditControlRequireStyle = "RequiredEdit"`
    a. The style to use for required edit controls. See the instructions above.
4. `BR.EditControlPlainStyle = "NormalEdit"`
    a. The style to use for edit controls that are not required. See the instructions above.
5. `BR.TheSecurityClass = SEC`
    a. If you are using the Security Class it should be initialized first, and this line lets the Business Rule class know that it can make calls into the security class.
6. BR.TheSystemLogClass = SysLog
    a. Pointer to the System Log class for logging errors. (*See Notes under Properties*)
7. BR.`SilentErrors = False`
    a. Optional If set to True Error Messages in the class will not be displayed. The default value is False. Note: If using the System Log class the error will still be logged even if an error is not displayed.
8. `BR.LoadRules()`
    a. Loads the Business Rules into an array so all future access is not database driven to improve performance.

Properties

Note: Private Properties and Members are not documented.

TheSecurityClass
Pointer to the Security Class for Field Level Security

- The Security class is another Open Source class available from NextAge. The class will perform without it. However if you don't not want to include the Security class in your project you will have to comment out the related lines of the class. Unfortunately WX does not provide the ability to optionally compile code based on the existence of another class. As long as the class is include in your project you will not have compile errors and the class calls will be skipped, however if you remove the Security class from your project, you will have compile errors in the Business Rules class.

EditControlPlainStyle
Style to be used for edit controls that are not required.

EditControlRequireStyle
Style to be used for required edit controls.

StyleSheet
The name of the style sheet that contains the EditControlPlainStyle and EditControlRequireStyle

SilentErrors

If set to True Error Messages in the class will not be displayed. The default value is False. Note: If using the System Log class the error will still be logged even if an error is not displayed.

TheSystemLogClass

Pointer to the System Log class for logging errors.

- The System Log class is another Open Source class available from NextAge. The Default Manager class will perform without it. However if you don't not want to include the System Log class in your project you will have to comment out the related lines of the class. Unfortunately WX does not provide the ability to optionally compile code based on the existence of another class. As long as the class is include in your project you will not have compile errors and the class calls will be skipped, however if you remove the System Log class from your project, you will have compile errors in the Default Manager class.

Methods

Note: Private methods are not documented.

AddControl
Tell the class to manage the business rules for the control. If you do not add the control then it will be completely ignore by the class and will not be displayed in the Business Rule Editor.

ApplyBusinessRules
Initializes a Windows and apply any display settings, such as changing the style of required fields and hiding or disabling fields based on the Field Level security.

DefaultRule
Sets up a default rule for a control. See detailed instructions above.

LoadRules
Loads all rules from the database into an array, to improve performance of the class.

ValidateAllFields
Force validates all controls for a window being managed by the class.

ValidateField
Validates a control being managed by the class. Can be called with soft validation or forced validation.

Code Bricks

BRAdd - Business Rules Add Control
Used to add a screen control to the Business Rule Manager, place in the *Initialization Process* of the control

BRApply - Business Rules Apply Business Rules
This will apply any formatting or field level security setting for any rules on the Window. It should be called from the window's *Initialization Process*

BRSetup - Business Rules Setup
Setups up the Business Rules class, should be in the Project Initialization code.

BRDefaultSec - Business Rules Setup Default Field Level Security
Setups up Field Level security for a control

BRDefaultReq - Business Rules Setup Default Required
This sets up a field as being required and displays the required style (Red Box) around the field

BRDefaultVal - Business Rules Setup Default With Validation
Includes custom validation code. See *Validation Code*

BRAllValidate - Business Rules Validate All Fields
Validates all Fields of the window, should be called prior to saving the record. Note this code brick has additional code related to the Browse Form Manager, if you are not using that class it can be removed.

BRValidate - Business Rules Validate Field
Performs the validation of the a field via the Business Rule Manager, place in the *Exit Event* of the field, for checkboxes place in the *Whenever Modified Event* instead.

Special Properties

The below properties can be used to change the default file name and field names used.

BusinessRulesFile

The name of the Business Rules file. Default = “BusinessRules”

BusinessRulesIDField
Default = "BusinessRulesID"

FieldNameField
Default = "FieldName"

BrushColorField
Default "BrushColor"

FontColorField
Default "FontColor"

RequiredField
Default "Required"

RequiredStyleField
Default "RequiredStyle"

CategoryField
Default = "Category"

AccessLevelField
Default = "AccessLevel"

StateField
Default = "State"

ValidationCodeField
Default = "ValidationCode"

The below Properties although public is intend for use by the Business Rule Editor Window and you should not need to use them directly.

ControlArray
An array that contains all of the information about the controls being managed by the class.

Special Methods

The below methods are public methods but they are intended to only be called by the Business Rules Editor Window or the Business Rule Editor Button Template, you should not need to call them directly in any other code.

DeleteBusinessRule
Completely deletes a rule from both the array and the database. Called from the delete button of the Business Rule Editor.

FetchBusinessRule
Retrieves a business rule from the array.

GetWindowName
Retrieves a the window name based on the number assigned to it by the class.

GetWindowNumber
Retrieves a the window number assigned to a window by the class. If the window has not been added to the class it will be added. Called from the Business Rule Editor Button templates initialization code, you should not need to call directly.

UpdateBusinessRule
Updates the business rule in both the array and the database. Called from the Business Rule Editor Window, you should not need to call directly.

Control Templates

Business Rule Editor Button

Adds the edit button to the screen and initializes the class for the window. Be sure to use the *Windows->Edit the Initialization Order* function and set the button as the first item to be initialized. It assumes that a local procedure called SetupDefaultRules exists. You must create this procedure even if you are not setting up any default rules for the window. The easiest way to do this is the Create a Base Window Template to be used for creating all your windows, and on this template include both the button template and the empty SetupDefaultRules procedure. The code of this template includes a call to the security class to hide the button if the user does not have security rights. You can change this call as required.

Windows

While not technically part of the class the below window is included in the NextAge Open Source Application that can be use as is or modified as you desire.

Win_BusinessRuleEditor
This window allows the business rules to be edited at runtime. The window is included in the NextAge Open Source application and can be imported into your application for this purpose. It can be reskinned and modified as you desire, however you should take great care that none of the logic that deals with the class is damaged. Note: the category combo uses a query to display all the possible categories, the queries used are declared in the NextAge Open Source application or you can create your own of course.

Change Log

1.0 - January 16, 2013

Initial Release