Security

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 visit www.thenextage.com or follow his blog at www.thenextage.com/wordpress

General Information

The Security Class has been around in one form or another since my very first project. At one time I provided a set of Clarion 3rd Party templates for it. The security class is not a replacement for groupware, or even meant to compete against groupware. I already had several applications in the wild with the using my security model and it was just easier for me to translate it to WX. The philosophy around the Security class is quite a bit different than groupware. The security class doesn't provide any real UI components, although there are sample windows in the NextAge Open Source application. Instead the security class is provides a foundation to configure default security at design time, and then have that security adjusted at runtime by a "superuser" without requiring additional development work. The premise is very simple, which is what actually allows it to be used for nearly anything. I have used it in an ERP system with 600+ users and not run into a security requirement that we could not cover.

The foundation of the security model is a numbered access level and a category. Each user is assigned a numeric "Global Access Level". Each user can also have Category override settings.

An Example:

Suzy is the AP clerk. She has a global access level of 3. She also has a category override level of 6 for AP, and 4 for GL.

1. The Sales Report is a category of Sales and requires as level 4.
2. The Customer Browse is a category of Client and requires a level of 3.
3. Paying an AP invoice is a category of AP and requires a level 6.
4. Generating a GL entry is a category of GL and requires a level of 3.
5. Dating a GL entry into a prior period is a category of GL and requires a 5.

When Suzy tries to:

1. Run the sales report, she is told she does not have access.
2. Run the Customer Browse she is allowed. Her global access is high enough
3. Pay an AP Invoice, she is allowed. Her global access is not high enough, but she has a category override that is.
4. When she generates the GL entry it is allowed, again her global access is high enough
5. If she tries to back date the entry, she is not allowed, her global access is not high enough and although she has a GL category override, it is not high enough either.

The security class is simple terms is a call to a "go - no go" function. It does all the work of testing the overrides, etc. and returns either a True or False.

Part of what makes the Security class powerful is that the setting are defaulted at design time but can be overwritten at runtime. This is accomplished by maintaining a "SecurityDetail" file.

This file has the Item being tested (be it a screen, function, whatever the developer decides to call it), a category, and an access level. All of that information is actual passed with the call to check the security. What the security class does, is checks the “SecurityDetail” file to see if the item already exist, if it does then the values from the file are used, otherwise a record is created in the “SecurityDetail” table with the default values supplied. This means that if the record in the “SecurityDetail” table is changed, it would override the defaults.

Back to our sales report example from #1 above. The first time some tried to run the sales report a record was written into the “SecurityDetail” table. With the item = “Sales Report”, Category = “Sales” and AccessLevel = 4. John the IT manager is told by the CEO that he wants anyone with at least a level 3 to run the sales report. John edits the “SecurityDetail” file and changes the AccessLevel to 3 on that record. Now when Suzy runs the Sales Report she has access because her global access level is now high enough.

As you can see the combination of Access Levels and Categories allows you to create as complex of a system as you want. It could even be dependant on more than one setting by simply making multiple calls to the class and not performing the function unless all of them pass.

Initial Setup

Since the class is designed as a “go / no go” function call, it doesn’t necessary take care of the logging in of a user, managing passwords, etc. However there are sample window for doing all of that provide with the NextAge Open Source Application. Below are the basic steps that should be followed to configure your application to use the Security class.

1. Instantiate the class in the Project Initialization code.
2. Require the user to log in
    a. I do this by making doing `Open(Win_Login)` in the Initialization Process of my first window.
    b. You could do it in your project code, or make active directory calls, or use your own custom login code, or follow Glenn Rathke example on how to use WX’s Groupware
    c. However you get the user “logged in” you need two pieces of information to pass to the security class
        i. User Id - This is the primary key to the User Table
        ii. Access Level - This is the user’s global access level.
3. Initialize the class, this call loads all of the security detail records, and user category override records into arrays. All future checking of security is done against the arrays making the class very efficient but also means that any changes to security will require the user to log off and back on for them to take effect. For our projects this has always proven to be a fair tradeoff since once security is setup it is rarely changed other than adding new users.
    a. `SEC.Initialize(`*`UserID,AccessLevel`*`)`
        i. UserID - is the primary key to the User Table
        ii. Access Level - is the users global access level

Checking Security

As mentioned the Security Class is designed as a “go / no go” function. A call to SecurityCheck returns a True or False from there the developer is responsible for what happens. Although this may seem limiting, it actually allows for quite a bit of flexibility in what you use the security class for. Remember that the act of calling the SecurityCheck method is also building the default SecurityDetail table. There are code templates to assist when checking the ability to open a window, run a report.

- SecurityCheck has four required and one optional parameter
    - ItemName
        - The unique identifier of the item to be tested, this could be a window name, a function name, whatever you desire.
    - Category
        - The default category for the item, this is only used to create the SecurityDetail record if it does not exists, the actual category tested will always be the value from the table.
    - Description
        - A description that will assist the Super User when configuring security. This value will be written into the SecurityDetail table if the record does not exist.
    - DefaultAccessLevel
        - The access level required to run the function. Again this is the default value that will be written to the SecurityDetail record if it does not exist, but the test will always use the value from the table.
    - DisplayMessage
        - An optional parameter that determines if the class should display a message telling the user they do not have access. This is useful when checking the permission for a function, as there is no coding needed by the developer for the message. But there are many times that you want the security check to be silent as all you are doing is hiding a field, etc.

Examples

- Testing access to the Customer Browse
    - ```
      IF NOT Sec.SecurityCheck (MyWindow..NAme , ”Client” , 3 , MyWindow..Title , True) THEN
          Close()
      END
      ```
    - If the user does not have at least Client access level 3 then they will receive an error message and the window will close
    - Notice the use of MyWindow..Name and MyWindow..Title to provide the item name and description
- Disable to the GL Entry Date control is the use does not have GL access level 5
      ```

- ```
  IF NOT Sec.SecurityCheck (“ChangeGLEntryDate” , ”GL” ,
  5 , ”Ability to Change the GL Entry Date” ) THEN
      EDT_EntryDate..Grayed = True
  END
  ```
- Notice this this call the display message parameter was not included, therefore the class did not display a message.
  ```

Checking Security for a Form

A special version of the SecurityCheck method that is used to check the users ability to Add, Change, and Delete for a form. It actually makes 3 calls to the SecurityCheck method concatenating “-Add”,”-Change” and “-Delete” to the ItemName passed in. A Code Brick is provided to assist with this *SecForm*

- SecurityCheckForm has nine parameters:
    - FormName
        - Three records will be added to the SecurityDetail table, the item name will be the Form Name with -Add, -Change, or -Delete appended to it. *NOTE: For internal windows you can not use the MyWindow reserved word, since it will return the “parent window”, not the “internal window” name.*
    - Category
        - The default category for the form, this is only used to create the SecurityDetail record if it does not exists, the actual category tested will always be the value from the table.
    - Description
        - A description that will assist the Super User when configuring security. This value will be written into the SecurityDetail table if the record does not exist.
    - AddLevel
        - The Access Level Required for Adds
    - ChangeLevel
        - The Access Level Required for Changes
    - DeleteLevel
        - The Access Level Required for Deletes
    - AddAllowed
        - Boolean Variable that will contain True or False once the security check has been made.
    - ChangeAllowed
        - Boolean Variable that will contain True or False once the security check has been made.
    - DeleteAllowed
        - Boolean Variable that will contain True or False once the security check has been made.
- After the call to the class has been performed the three variables passed will contain true or false indicating if the action is allowed and can be used in your code as needed.

Database Tables

There are two tables required for the Security class. The “SecurityDetail” and “SecurityCategory” tables. Technically you also need to maintain users and their global access level, but the class itself doesn’t need to know about that table, only the value of the UserId and global access level of the user. The class has been written in a generic fashion that allows the file names 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 the file follows. If you do not use these definition you will have to use the associated properties to point to the correct values (*See the Special Properties*).

SecurityDetail
Holds the requirements for each Window, procedure, function, etc that requires a security check.

- SecurityDetailID
  - Automatic ID
- ItemName - Unique Identifier of the Item to test.
  - Text (50)
- AccessLevel - Access Level Required
  - Integer
- Category - Security Category of the Item
  - Text (20)
- Description - Description to assist user maintaining the security
  - Text (50)

SecurityCategory
Category Override records for a user.

- SecurityCategoryID
  - Automatic ID
- UserID- The Unique Identifier for a User
  - Integer
- Category - Security Category of the Override
  - Text (20)
- AccessLevel - Access Level of the Override
  - Integer

Project Code

The class should be instantiated in the project Intialization code, since one instance of the class will be used for the entire application. If you are using the special properties to override the file and field names those properties should also be set here.

1. `SEC is Security`
    - Instantiates an instance of the class global for the entire project.
2. `SEC.SilentErrors = False`
    - 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
3. `Sec.TheSystemLogClass = SysLog`
    - Pointer to the System Log class for logging errors. (*See Notes under Properties*)

Properties

Note: Private Properties are not documented.

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 Security 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 Security class.

Methods

Note: Private Methods are not documented.

BusinessRuleSecurityCheck
A special method that is used by the Business Rule manager class for field level security via the Business Rule Manager class, this method is not meant to be called directly.

Initialize
Initialize the class, this call loads all of the security detail records, and user category override records into arrays. All future checking of security is done against the arrays making the class very efficient but also means that any changes to security will require the user to log off and back on for them to take effect. For our projects this has always proven to be a fair tradeoff since once security is setup it is rarely changed other than adding new users.

SecurityCheck
As mentioned the Security Class is designed as a “go / no go” function. A call to SecurityCheck returns a True or False from there the developer is responsible for what happens. Although this may seem limiting, it actually allows for quite a bit of flexibility in what you use the security class for. Remember that the act of calling the SecurityCheck method is also building the default SecurityDetail table. There are code templates to assist when checking the ability to open a window, run a report.

SecurityCheckForm
A special version of the SecurityCheck method that is used to check the users ability to Add, Change, and Delete for a form. It actually makes 3 calls to the SecurityCheck method concatenating “-Add”,”-Change” and “-Delete” to the ItemName passed in. A Code Brick is provided to assist with this *SecForm*

Code Bricks

SecCheck
Security Checking for a Window. It uses the properties MyWindow..Name and MyWindow..Title for the ItemName and Description

SecForm - Security Checking for Forms
Security Checking for a form. It defines the the boolean variables needed. It uses MyWindow..Name and MyWindow..Title for the ItemName and description, there is a optional call that is commented out that uses {MySelf..Name,indControl}..SourceWindow, and the caption of a table to provide the ItemName and Description which can be used when the form is an internal window.

There is also code to disable the Add, Change and Delete buttons based on the boolean variables, there names of the buttons match the FormButtons Control template from the NextAge Open Source Application. You can of course change the brick to meet your needs.

SecRpt - Security Checking for Reports
Security Chechecking for a report. It uses MyReport..Name for the ItemName.

Special Properties

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

SecurityDetailFile
Holds the requirements for each Window, procedure, function, etc that requires a security check.

SecurityDetailItemNameField
Unique Identifier of the Item to test.

SecurityDetailAccessLevelField
Access Level Required

SecurityDetailCategoryField
Security Category of the Item

SecurityDetailDescriptionField
Description to assist user maintaining the security

SecurityCategoryFile
Category Override records for a user.

SecurityCategoryUserIdField
The Unique Identifier for a User

SecurityCategoryCategoryField
Security Category of the Override

SecurityCategoryAccessLevelField
Access Level of the Override

Windows

While not technically part of the class there are a few windows included in the NextAge Open Source Application that can be use as is or modified as you desire.

Win_Login
Called from first window of the application, to require a user to log in. Does not actually make any calls into the class. If the login is successful, a global variables are set for UserId and Global Access Level which are later used with calls to the Security Class.

Win_ChangePassword
Allows the user to change their password, called from the login screen, also called from user edit screen. It uses the global variable for the salt phrase, but there is nothing “class” specific about the window itself.

IBRW_User
Internal Window that uses the Browse Form Window Manager to provide a User Browse with a Peek-A-Boo Edit Form with both the user and the category overrides.

Change Log

1.0 - January 12, 2013

Initial Release

1.01 - January 16, 2013

- Fixed issue with Array Sort Error when adding new entry