Security Namespace Reference

Access security functions via scripts.

Examples on Integration of the Security Module with scripts.

- Examples on Integration of the Security Module with scripts.
- Managing Users with Scripts
- Customizing Login Procedures
- Monitoring Client Connections

Managing Users with Scripts

During the launch of an application, the Login process accepts all users defined in the Solution Configuration. Additionally, it is possible to add Runtime Users during runtime execution dynamically, and they are stored in a default database located in **Datasets / BDs / RuntimeUsers**. Customization of the database used for user storage is also possible in the Module Datasets. Furthermore, users can be created or modified using the Security Namespace's available methods.

To simplify user management, a platform Plugin is available in the Security module that provides a template application for creating a Runtime Display. The UserManagement Plugin can be imported by selecting the appropriate Plugin in **Solution / Import Plugins**. Once imported, the Plugin creates names, such as `SecurityAccounts` and `ChangePass`, that aid in user management.

The Security Namespace offers a wide range of methods that can be used to create or modify Runtime Users before starting runtime execution or during project execution. The documentation provides detailed information on the available features and methods, allowing users to enhance their application's functionality and flexibility during runtime. By utilizing the Security module's capabilities, users can effectively manage Runtime Users, increasing the overall security and functionality of their applications.

In the following, you can find more details regarding the available RuntimeUser methods in the Security namespace:

In application management, the dynamic addition of users during runtime is an essential capability that enhances flexibility and control over user accessibility. The **AddRuntimeUser** method in the Security namespace is a powerful function designed for this purpose.

This function adds a new runtime user with detailed information. By accepting various parameters, it allows for a detailed definition of each user, including their identity, permissions, policy, contact information, and security settings.

AddRuntimeUser

```
@Security.AddRuntimeUser(string name, string permissionsStr, string password, string passwordHint, string policyStr, string profilePhone, string profileCompleteName, bool oneTimePassword)
```

The `name` parameter refers to the user's name, providing a unique identifier for each user added during runtime. It creates the fundamental layer of user identification, offering an easy way to distinguish one user from another.

The `permissionsStr` parameter is a string that defines the permissions allocated to the user. This parameter plays a pivotal role in determining what the user can access and modify within the application, providing a means of control and security by limiting user actions according to their assigned permissions.

The `password` parameter is self-explanatory, serving as the key to user authentication. Coupled with the `passwordHint`, these parameters form the basis of security for each user, ensuring that only authorized individuals gain access to the allocated permissions and data.

The `policyStr` parameter represents the policy associated with the user. This string offers another layer of control over user actions by defining the rules and guidelines that the user must adhere to within the application.

The `profilePhone` and `profileCompleteName` parameters provide additional information about the user, such as their phone number and full name. This information can be useful for various purposes, including user identification, contact, and personalization of the user experience.

The `oneTimePassword` parameter is a boolean value that defines whether the user needs to change their password after their first login. This feature enhances security by ensuring that even if an unauthorized person initially sets up the user account, the legitimate user has the chance to secure their account at their first login.

The `NewRuntimeUser` method also creates a new runtime user. You pass the user's name as a string to the name parameter. If there's an error during the creation of the user, an error code will be outputted via the errorCode parameter.

NewRuntimeUser

```
@Security.NewRuntimeUser(string name, out int errorCode)
```

Customizing Login Procedures

The Login page can be edited to modify its layout. To do this, select the LogOn display. In the code behind that display, the method **Security.Logon()** is used for user validation. If you wish to implement a different user validation system, you can modify the logic and call your validation system before invoking the LogOn method. The LogOn method should be called based on the results of your validation.

Another option to customize the logon process is to add your custom logic to the **ClientStartup** script task. This script runs on any computer connected to the server application. You can perform checks based on criteria such as computer IP, computer name, Windows Active Directory logged user, or any other relevant factor to determine if the user is authorized to start the application and specify their credentials. If unauthorized access is detected, you can terminate the application by calling **Client.Shutdown**. Alternatively, if the user is allowed access, you can call **Security.Logon()** for a user corresponding to the selected security profile.

By default, when launching the application on a client computer, it starts with the user **GUEST** instead of requesting a login. The GUEST user is similar to an anonymous login. If you do not want this behavior in your application, you can replace the startup page with a page that prompts for logon information.

Monitoring Client Connections

The Designer tool, at Security SecurityMonitor, when the solution is running, there is a list of client processes connected with the server.

That information is also available for the scripts, with the method: Server.GetAllConnections()

For advanced diagnostics during the Design phase, the tool ModuleInformation.exe has a section with all the TCP/IP connections.

In this section...