Step-by-Step Guide

CelloSaaS Business Edition Integration

CelloSaaS Version: 4.3.2.0

The following are the steps for getting started with the Integration of CelloSaaS Business Edition

1. Open up the application solution folder
2. Create a folder and call it as “Thirdpartydlls”, extract the dll files in the attached “thirdpartydlls.zip” file to this folder.
3. Launch Visual Studio 2012 and then open the application.
4. In the solution's web application, add reference to the above dll's.
5. Open the “web.config” file of the web application in Visual Studio.
6. Navigate to the <configSections> element and add the following section

   ```
   <section name="IdentityProvider" type="CelloSaaS.BusinessEdition.IdentityProviderConfiguration, CelloSaaS.BusinessEdition"/>
   ```

7. Add the following IdentityProvider section in the web.config files

```
<IdentityProvider defaultProvider="Default">
       <providers>
              <add name="Default" assembly="CelloSaaS.BusinessEdition"
type="CelloSaaS.BusinessEdition.CelloIdentityProvider"/>
       </providers>
</IdentityProvider>
```

9. In the web.config file, under <appSettings> section and then add the following key-value mapping

```
<appSettings>
       <add key="AdminUrl" value="http://admin.application.com/"/>
       <add key="CelloCookieName" value="CelloCookie"/>
       <add key="CelloCookieSalt" value="company"/>
       <add key="UseAPIKey" value="true"/>
       <add key="CommonApiKey" value="UBd2DX9xIPprqtB7D0yb0w=="/>
       <add key="ProductAdminWCFApiKey" value="UBd2DX9xIPprqtB7D0yb0w=="/>
       <add key="ProductAdminUserName" value="admin@company.com" />
       <add key="UpgradeSubscriptionUrl"
value="http://admin.application.com/MySettings/ManageSettings"/>
       <add key="UpdatePaymentAccountUrl" value="http://admin.
application.com/Billing/ManagePaymentAccount"/>
       <add key="ApplicationModules" value=""/>
</appSettings>
```

10. Update the Wcf services configuration’s AppSettings.Config file with the following keys to use the common API key for Service calls

    ```
       <add key="UseAPIKey" value="true"/>
       <add key="CommonApiKey" value="UBd2DX9xIPprqtB7D0yb0w=="/>
    ```

12. Add the web service configurations to the web.config file. The complete configurations section is attached herewith in the enclosed in the following files “clients.config” and “bindings.config” file. The corresponding service URL or IP address change has to be made in the clients.config file.
13. Add a folder in the web application and call it as “config” and place the files “bindings.config” & “clients.config” inside that folder.

Business Edition API & Helpers How Tos

Required namespaces

```
using CelloSaaS.BusinessEdition;
using CelloSaaS.BusinessEdition.ServiceProxies.AccessControlManagement;
using CelloSaaS.BusinessEdition.ServiceProxies.LicenseManagement;
using CelloSaaS.BusinessEdition.ServiceProxies.LicenseService;
using CelloSaaS.BusinessEdition.ServiceProxies.TenantService;
using CelloSaaS.BusinessEdition.ServiceProxies.UserDetailsService;
using CelloSaaS.BusinessEdition.Utilities;
```

I) How to Authenticate

In order to authenticate the user, the following API call will be made from the business application.

Method to use:

```
AuthenticationStatus status = AuthenticationHelper.Login(userName,password, null,rememberMe);
```

Parameters:

Username: The unique username that is registered against a tenant

Password: The login password

CompanyCode: The unique company code to identify the tenant that the user belongs to

PersistCookie: To indicate whether the cookie is to be persisted. This is optional. Default value is false

RaiseEvent: To indicate whether to raise a login event once the login operation has succeeded. This is optional. Default value is true.

Return:

This API call returns the AuthenticationStatus enumeration which has the following enumeration values.

Enum Value	Indication
`FirstTimeUser`	The user is logging into the application for the first time. This user has to be re-directed to the CelloSaaS.BusinessEdition application to fill in the security question and answer.
`InvalidTenantLicense`	The tenant license is invalid
`InvalidUserOrPassword`	The user name or the password is invalid
`NoRoles`	The user logging in has no roles assigned
`PasswordChangeForced`	User to be redirected to the password change page as the user password change has been forced by the tenant administrator.
`PasswordExpired`	User password has expired
`PasswordPenultimateFailureAttempt`	The number of attempts for the password has reached the penultimate attempt

`Success`	The login attempt succeeded
`TenantLocked`	The tenant is locked out and hence the tenant and the users cannot be logged in
`TenantUnApproved`	The tenant is not yet approved.

II) How to check for tenant license

III) How to get the UserDetails

IV) How to get the Roles for the Logged-in user

V) How to get the Privileges for the logged-in user

VI) How to check for the privileges

Method to use

AuthenticationStatus status = AuthenticationHelper.Login(userName,password, null,rememberMe);

Parameters:

Return:

Sample:

```
AuthenticationStatus status = AuthenticationHelper.Login(userName,password, null,rememberMe);

switch (status)
{
    case AuthenticationStatus.FirstTimeUser:
// Take some action for the first time user
        break;
    case AuthenticationStatus.InvalidTenantLicense:
// Indicate to the tenant that the license is invalid
        break;
    case AuthenticationStatus.InvalidUserOrPassword:
// Invalid user name or password
        break;
    case AuthenticationStatus.NoRoles:
// The user has no roles assigned.
        break;
    case AuthenticationStatus.PasswordChangeForced:
// The user password change has been forced.
        break;
    case AuthenticationStatus.PasswordExpired:
// The user password change has expired.
        break;
    case AuthenticationStatus.PasswordPenultimateFailureAttempt:
// The currently provided password has reached the penultimate attempt.
        break;
    case AuthenticationStatus.Success:
```

```
// The user login has succeded
        break;
    case AuthenticationStatus.TenantLocked:
// The tenant has been locked. Contact the product administrator.
        break;
    case AuthenticationStatus.TenantUnApproved:
// The tenant is not yet approved. Contact the product administrator.
        break;
    case AuthenticationStatus.Unknown:
// The Authentication status is unknown, the exception details has to be verified.
        break;
    case AuthenticationStatus.UserDeactivated:
// The user is currently de-activated
        break;
    case AuthenticationStatus.UserLocked:
// The user account is currently locked
        break;
    default:
        break;

}
```

The AuthenticationStatus should be "Success" for the successful login, for all the other options in this enumeration, report a failure message to the end user based on the corresponding error code returned from the library as shown in the above enumeration switch case block of statements.

This authentication process will create a cookie called as “CelloCookie” as per the configured cookie name in the AppSettings section. This cookie is independent of the Authentication Cookie that the application may be set. This is used mainly to set the UserIdentity Context to be used by Cello for WCF service invocation. The application can also make use of the properties in UserIdentity [User Context].

VII) Obtaining the User, Role and Privilege Details

The following will be the API calls that can get the user details by email id, the roles and the user’s privilege [aggregate of the roles]

```
// To Get the userdetails based on the email id
UserDetails userData = UserDetailsProxy. GetUserDetailsByEmailId(model.UserName);

if (userData != null && userData.MembershipDetails != null &&
!string.IsNullOrEmpty(userData.MembershipDetails.TenantCode))
{
       // To Get the Tenant Id of the User based on the above obtained data
       string tenantId = userData.MembershipDetails.TenantCode;

       // To Get the User Roles of the User based on the tenant id
        string[] userRoles = RoleProxy.GetUserRolesForTenant(userData.User.Identifier, tenantId,
tenantId);
        string[] userRoles = RoleProxy.GetUserRolesForTenant(userData.User.Identifier,
tenantId, tenantId);
       if (userRoles != null && userRoles.Length > 0)
       {
         // To Get the privileges for the logged in user for the tenant and the roles
           List<string> userPrivileges = PrivilegeProxy.GetPrivilegeNames(tenantId,
       userRoles);
       }
}
```

VIII) PrivilegeHelper

The privilege helper provides the following API to be consumed by the application

1. GetApplicationSpecificPrivileges
    1. Gets the list of privileges that the currently logged in user possesses based on the application modules
    2. There are two API exposed, one will obtain the comma-delimited module codes from the AppSettings and then fetches the privileges based on the modules
    3. The overloaded method takes an array of module codes and returns the privileges based on the given module codes.
2. GetAllPrivileges
    1. This API call will return the consolidated [CelloSaaS + Application] privilege for the logged in user, based on the licensed modules
3. CheckAllPermissions
    1. This is a web service call
    2. Input is an array of privileges
    3. Check if the logged in user has all of the given privileges and returns the result array in the same order
4. CheckPermissions
    1. This is a web service call
    2. Input is an array of privileges
    3. Check if the logged in user has the given privileges and then returns the Boolean array indicating the status of the privilege check.
5. CheckPermission
    1. This is a web service call
    2. Input is a privilege
    3. Check if the logged in user has the given
6. Related AppSettings to set the application specific modules comma-delimited.
    1. `<add key="ApplicationModules" value="AppModuleNames"/>`

IX) TenantLicenseValidator

1. CheckValidity
    1. This API call takes a single parameter called as "enableAutoUrlRedirection" which is a nullable Boolean parameter which indicates that the URL redirection has to be taken care within the library based on the configured URL in the AppSettings section of the web.config file.
    2. In case this parameter is set as true, this API call will return any one of the following enumeration value from LicenseValidationStatus.

    - UpgradeSubscription : To redirect the tenant admin to the subscription upgradation page
    - UpdatePaymentAccount: To redirect the tenant admin to the Payment update page
    - None: No action is required

    3. Related Appsettings to capture the Subscription or the payment URL

```
<add key="UpgradeSubscriptionUrl" value="http://admin.app.com/MySettings/ManageSettings" />
```

```
<add key="UpdatePaymentAccountUrl" value="http://admin.app.com/Billing/ManagePaymentAccount"
/>
```

Sample Code

X) Tenant License Validity Inspection

```
var licenseValidityStatus = TenantLicenseValidator.CheckValidity();

if (licenseValidityStatus != LicenseValidationStatus.None)
{
    return RedirectToAction("Logon", "Account", new { @message = licenseValidityStatus });

}
```

(OR)

```
TenantLicenseValidator.CheckValidity(true);
```

XI) Obtaining Privileges

```
a. Obtaining the privileges pertaining to the application modules
var appPrivileges = PrivilegeHelper.ApplicationPrivileges();

if (appPrivileges == null || appPrivileges.Count < 1)
{
    // User Does not have any privileges
}
else
{
    // There are some application specific privileges for the user
}

b. Obtaining the consolidated privileges
// Gets the complete list of privileges
List<string> consolidatedPrivileges = PrivilegeHelper.Privileges();
```

XII) Pre & Post Processors

The following are the steps to create Tenant Pre and Post processors to extend the Tenant Management functionality provided by Cello.

1. Create a new Class library project targeting .Net Framework version 4.5
2. Add the library files to this project namely
    1. CelloSaaS.Services
    2. CelloSaaS.ServiceContracts
    3. CelloSaaS.Model
    4. CelloSaaS.Library
3. Create a class and call it as “TenantProcessors”
4. Implement the following interfaces in this class
    1. IPreProcessorProvider [Pre-Processing]
    2. IPostProcessorProvider [Post-Processing]
5. Implement the interface members in this class

6. Inside the method definition's that are not required to be used, simply return true so that it continues to bubble up the processing in the service pipeline
7. The following are the parameters available in the members
    1. referenceId: The primary key value [Tenant Identifier – Guid Value as string]
    2. entity: The primary entity [Tenant for PreProcessorInsert and TenantDetails for PostProcessorInsert
    3. Wrap the entire member body implementation in a try … catch … code block and then place a "return false;" statement inside the catch block after handling the exception if required to indicate the service pipeline to terminate and rollback the transaction.
8. Cast the entity as the appropriate type and then process the data and then end the process with a "return true;" statement so that the service pipeline execution proceeds and then completes the transaction scope.
    1. `TenantDetails tenantDetails = entity as TenantDetails;`
9. Once the processors are completed, build the library in "Release" mode and then add this library to the "CelloSaaSDlls" folder of the WebApplication / Admin provisioning application
10. Edit the web.config file to register the processors as shown below.
11. Add the section "`processors`" to the `<configSections>` in the web.config file

```
<section name="processors"
type="CelloSaaS.Services.ProcessorConfiguration,CelloSaaS.Services"/>
```

12. Create section "processors" and then specify the corresponding pre and post processor type and assembly registrations as given below

```
<processors>
       <preprocessors>
              <add name="TenantPreProcessor" assembly="TenantServiceHooks"
type="TenantServiceHooks.TenantServiceExtensions" />
       </preprocessors>
       <postprocessors>
              <add name="TenantPostProcessor" assembly="TenantServiceHooks"
type="TenantServiceHooks.TenantServiceExtensions" />
       </postprocessors>

</processors>
```

13. Now, add reference to the newly added pre and post processor libraries to the web application project references.
14. Rebuild the entire solution
15. Execute the application and then create the tenant and verify the execution results of the pre and post processors.

XIII) Logout Process

The following sample code can be used to logoff from the Cello Context and the application context via the API. In the AuthenticationHelper.LogOut call, there are two default parameters

a. formAuthenticationSignout [default:false] - To indicate that the formauthenication can be signed off
b. raiseEvent[default: true] – To indicate that the logoff operation raises an event to register the logoff operation.

Sample Code

```
        AuthenticationHelper.LogOut(true);
```

Contact Information

Any problem using this guide (or) using Cello Framework. Please feel free to contact us, we will be happy to assist you in getting started with Cello.

Email: support@techcello.com

Phone: +1(609)503-7163

Skype: techcello