Cello How-To Guide

Checklist: Creating Modules and Entities

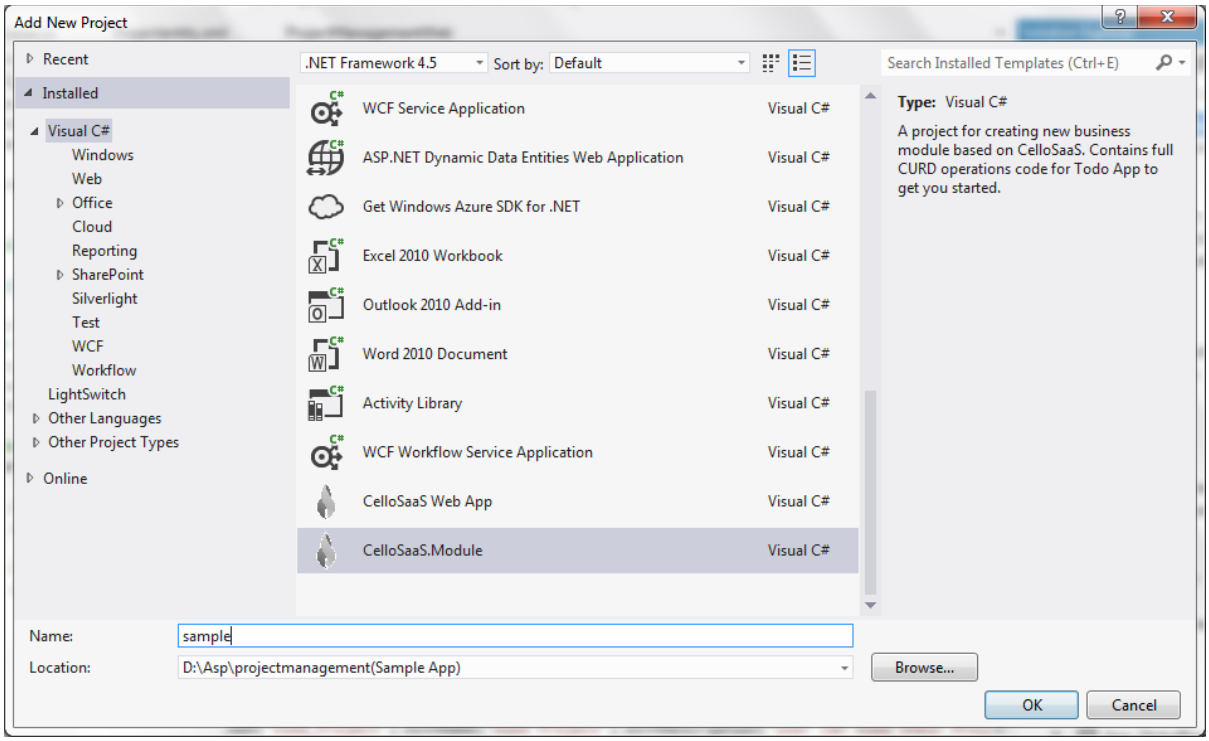
techcello

Contents

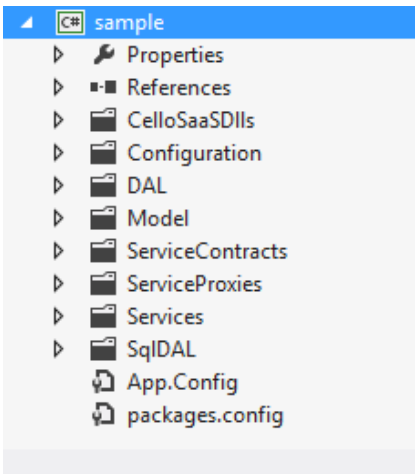
1. Introduction:

Developers can get started with their business modules using the Techcello module. This module provides a template of all the files required inside the business module.

To add a new techcello module add a new project to your solution which contains the CelloSaaS.webapp. Select the CelloSaaS.module template and create the module in parallel with your solution.



A sample module template will contain the following folders:



Each of the folders contain specific files

Configuration:

This folder contains Fluent API level configuration of the Modules, Entities & DataViews.

DAL & ServiceContracts:

The Data Access and Service layers are to be created using the interface driven approach to facilitate the loose coupling and also to have pluggable architecture in place

Model:

The model represents the business entities of the application.

ServiceProxies:

The service proxy is used a generic interface layer to the application so that this layer can make the in-prog calls or WCF calls and the application does not change the way in which it interacts with the application.

Services:

Concrete implementation of Service Contract

SqlDAL

Concrete implementation of DAL

Even though this module has been created it has to be configured inside the CelloSaaS Web App which has the cello modules.

NOTE:

It is not necessary that the business module must be generated from the code template techcello provides. Developers can create their own modules and follow the steps mentioned in the later sections of this document

2. Creating and Registering Entities inside modules:

Refer POCO document for entity creation in techcello

Once an entity is created it has to be noted that

- The entity is inherited from TenantTransactionalEntity and applied necessary attributes.
- Your DbContext class is inherited from CelloDbContext which takes care of data partitioning & security. That is routing the queries to the database according to the logged in tenant configuration. It also applies auto tenant isolation for your select queries.

```
    public partial class ProjectManagementModelContext : CelloDbContext
    {
        /// <summary>
        /// Creates Db Context with the given connection string name and metadata
        /// </summary>
        /// <param name="connectionStringName">connection string name e.g.: ApplicationConnectionString</param>
        /// <param name="metadata">metadata url</param>
        /// <param name="privilege">privilege</param>
        /// <param name="entityIdentifier">entityIdentifier</param>
        /// <param name="applyTenantScope">applyTenantScope</param>
        /// <param name="autoTenantIsolation">autoTenantIsolation</param>
        /// <param name="elevatePrivilege">elevatePrivilege</param>
        public ProjectManagementModelContext(string connectionStringName, string metadata = "res://*/", string
privilege = null, string entityIdentifier = null, bool applyTenantScope = false, bool autoTenantIsolation = true, bool
elevatePrivilege = false)
            : base(connectionStringName, metadata, privilege, entityIdentifier, applyTenantScope, autoTenantIsolation,
elevatePrivilege)
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
        }

        public virtual DbSet<EmployeeTask> EmployeeTasks { get; set; }
        public virtual IQueryable<EmployeeTask> EmployeeTasksQuery { get { return
this.ApplySecurity<EmployeeTask>(this.EmployeeTasks); } }
        public virtual DbSet<Project> Projects { get; set; }
        public virtual IQueryable<Project> ProjectsQuery { get { return this.ApplySecurity<Project>(this.Projects); } }
        public virtual DbSet<ProjectTask> ProjectTasks { get; set; }
        public virtual IQueryable<ProjectTask> ProjectTasksQuery { get { return
this.ApplySecurity<ProjectTask>(this.ProjectTasks); } }
    }
```

Above class is auto generated by Techcello POCO Generator templates.

In your DAL Implementation class:

```
private const string _metadata =
"res://*/Model.HelloWorldModel.csdl|res://*/Model.HelloWorldModel.ssdl|res://*/Model.HelloWorldModel.msl";
private const string ConnectionStringName = "HelloWorldConnectionString";

protected override string DoCreate(DataCreateRequest dataCreateRequest)
        {
            var entity = dataCreateRequest.Entity as Model.Todo;

            using (var ctx = GetContext())
            {
                ctx.Todoes.Add(entity);
                ctx.SaveChanges();
                entity.Identifier = entity.Id.ToString();
                return entity.Identifier;
            }
        }
       protected override Dictionary<string, Model.Todo> DoSearch(DataSearchRequest dataSearchRequest)
        {
            var condition = dataSearchRequest.SearchCondition as TodoSearchCondition;

            using (var ctx = GetContext())
            {
                var query = ctx.TodoesQuery; // This will apply auto tenant filter

                if (!string.IsNullOrEmpty(condition.TaskName))
                {
                    query = query.Where(x => x.TaskName.Contains(condition.TaskName));
                }
```

Create an entity using the POCO generator. Refer: Entity Framework POCO Generator Plugin

The POCO generator serves two purposes:

1. Replaces the context files to include datascoping capabilities.
2. Adds configuration details for the Entities

Notes to remember:-

- Your Entity must contain Primary Key named "**Id**".
- Your Entity must contain one column named "**TenantId**".
- Use ***DbContext.EntityNames*** for Create, Update and Delete operations
- Use ***DbContext.EntityNamesQuery*** for select operations.

Once an entity is created, create a dataview object related to the entity. Refer how_to_celloform_&_cellogrid_customization.

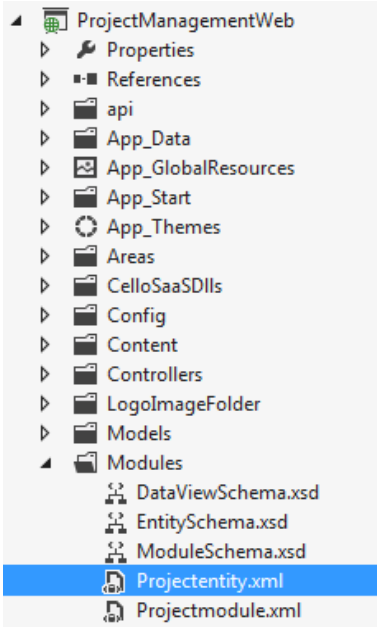
Registering an entity

Registering an entity into your module can be done in two ways:

1. Xml mode
2. Fluent API

XML mode:

To register all entities of a module in Techcello using XML's the xml name should end with entity. And should be placed under Cello Webapp >> Modules



The privileges for each of the entity and the table columns must be listed in the xml.

```
<?xml version="1.0" encoding="utf-8" ?>
<EntityConfiguration>
  <Entities>
    <!--cellohelp:entityconfiguration-->
    <Entity Identifier="Project" EntityName="Project" PrimaryKeyName="Id" IsExtensible="true"
TenantIdColumnName="TenantId" SchemaTableName="dbo.Project"  DisplayColumnName = "ProjectName"
SchemaTableConnectionStringName="ProjectModuleConnectionString"
ExtnSchemaTableConnectionStringName="ProjectModuleConnectionString" ExtnSchemaTableName="dbo.ProjectExtn">
      <Privileges>
        <Privilege>View_Project</Privilege>
        <Privilege>Add_Project</Privilege>
        <Privilege>Delete_Project</Privilege>
        <Privilege>Edit_Project</Privilege>
        <Privilege>Search_Project</Privilege>
      </Privileges>
      <Fields>
        <Field Identifier="Id" IsUnique="true"  Name="Id" TypeID="Guid"/>
        <Field Identifier="TenantId" IsUnique="false"  Name="TenantId" TypeID="Guid"/>
        <Field Identifier="ProjectName" IsUnique="true" Name="Project Name" TypeID="string"/>
        <Field Identifier="ProjectDescription" IsUnique="false" Name="Project Description" TypeID="string"/>
        <Field Identifier="StartDate" IsUnique="false" Name="Start Date" TypeID="DateTime"/>
        <Field Identifier="EndDate" IsUnique="false" Name="End Date" TypeID="DateTime"/>
        <Field Identifier="Project_Status" IsUnique="false" Name="Project Status" TypeID="bool"/>
        <Field Identifier="CreatedBy" IsUnique="false" Name="Created By" TypeID="Guid"/>
        <Field Identifier="CreatedOn" IsUnique="false" Name="Created On" TypeID="DateTime"/>
        <Field Identifier="UpdatedBy" IsUnique="false" Name="Updated By" TypeID="Guid"/>
        <Field Identifier="UpdatedOn" IsUnique="false" Name="Updated On" TypeID="DateTime?"/>
        <Field Identifier="ProjectOwner" IsUnique="false" Name="Project Owner" TypeID="string"/>
        <Field Identifier="ProjectManager" IsUnique="false" Name="Project Manager" TypeID="string"/>
        <Field Identifier="Comments" IsUnique="false" Name="Comments" TypeID="string"/>
        <Field Identifier="Cost" IsUnique="false" Name="Cost" TypeID="double"/>
        <Field Identifier="ProjectPriority" IsUnique="false" Name="Project Priority" TypeID="string"/>
        <Field Identifier="Type" IsUnique="false" Name="Project Type" TypeID="string" PickupListId="06583C29-
9B73-E311-A964-000C29C8E241"/>
      </Fields>
    </Entity>
  </Entities>
</EntityConfiguration>
```

Once an XML has been created. make sure that the following line is available in the Web APP>> Global.asax file under the `Configure()`method

```
CelloSaaS.Configuration.CelloConfigurator.RegisterEntity<CelloSaaS.Configuration.XmlEntityConfigurator>();
```

Fluent API:

Open Configuration>>ModuleConfig.cs file in your new module. Create a class ending with the name EntityConfigurator, inheriting IEntityConfigurator class. Under its Configure (EntityConfig) method configure the business entity.

```
public class TodoEntityConfigurator : IEntityConfigurator
    {
        /// <summary>
        /// Register your business entities details with CelloSaaS.
        /// </summary>
        /// <param name="entityConfiguration">The entity configuration.</param>
        public void Configure(EntityConfig entityConfiguration)
        {
            entityConfiguration.AddFromAssemblyOf<Model.Todo>(); // rename "Todo" with any one of your business entity
        }
    }
```

Once Config.cs has been updated, it has to be configured in the Web APP>> Global.asax file under the `Configure()`method.

```
CelloSaaS.Configuration.CelloConfigurator.RegisterEntity<Modluename.TodiEntityConfigurator>();
```

Attributes of an entity Model:

Attribute	Description
PrimaryKeyName	This property identifies the primary key column of the entity. This field will represent the primary key constraint in the database table.
IsExtensible	Marking this property as true enables this entity to have extended fields. Further to this setting [true], the ExtnSchemaTableName and ExtnSchemaTableConnectionStringName properties are to be filled with the appropriate values.
SchemaTableName	The table which will store the values for the base fields of the entity.
ExtnSchemaTableName	The table which will store the extension field values for the entity's extension fields.
SchemaTableConnectionStringName	The connection string name to identify the connection string for the data base that will contain the entity data
ExtnSchemaTableConnectionStringName	The connection string name to identify the connection string for the data base that will contain the entity's extension field data
Privileges	The comma delimited string of privileges. This is the list of privileges that are associated to the entity. These privileges will be demanded when accessing the entity from the service layer.
Features	The comma delimited string of features which will contain the features that this entity associates to.

If you intend to use extension tables, create extended table using the cello extension script and enable "is extensible" in your entity. Refer how_to_custom_fields.

3. Registering a module inside the Web App:

A new module can be registered in the CelloSaaS web app in two ways

1. Xml mode
2. Fluent API mode

XML mode:

An XML configuration has to be created under webapp>>module folder with its name ending in "module.xml" ex: projectmodule.xml. The xml configuration file should contain the details of all the modules, features, privileges and the entities inside the modules.

If there are any usage based components inside the module ex: User, Projects , they have to be mentioned in the XML configuration file.

```
ModuleConfiguration>
  <Modules>
    <!--cellohelp:moduleconfiguration-->
    <Module Code="ProjectModule" Name="Project Module" Description="This module contains feature and usgae of
the project">
      <!--cellohelp:featureconfiguration-->
      <Features>
        <Feature Code="ProjectFeature" Name="Project Feature">
          <Privileges>
            <Privilege Name="Search Project" Description="user can search their Project
list">Search_Project</Privilege>
            <Privilege Name="View Project" Description="user can View their Project
list">View_Project</Privilege>
            <Privilege Name="Add Project" Description="user can add their Project.">Add_Project</Privilege>
            <Privilege Name="Edit Project" Description="user can Edit their Project.">Edit_Project</Privilege>
            <Privilege Name="Delete Project" Description="user can delete their
Project.">Delete_Project</Privilege>
            <Privilege Name="Approve Project" Description="user can Approve the
Project.">Approval_Project</Privilege>
          </Privileges>
          <Entities>
            <Entity>Project</Entity>
          </Entities>
        </Feature>
        <!--cellohelp:usageconfiguration-->
      <Usages>
        <Usage Code="ProjectUsage">
          <Name>Project Usage</Name>
          <TypeCode>BLOCKUSAGE</TypeCode>
          <TypeName>BLOCKUSAGE</TypeName>
          <Threshold>10</Threshold>
        <CanGenerateInvoice>false</CanGenerateInvoice>
        </Usage>
      </Usages>
    </Module>
  </Modules>
</ModuleConfiguration>
```

Once an XML configuration file is created, make sure that the following line is available in the Web APP>> Global.asax file under the `Configure()`method.

```
CelloSaaS.Configuration.CelloConfigurator.RegisterModule<CelloSaaS.Configuration.XmlModuleConfigurator>();
```

With this the module is configured in Webapp by the XML mode

Fluent API mode:

Open Configuration>>ModuleConfig.cs file in your new module. Create a class ending with the name ModuleConfigurator, inheriting IModuleConfigurator class. Under its Configure(ModuleConfig) method configure the module, the usages and entities related to the module, the features available inside the module along with the privileges related to them.

```
public class ProjectModuleConfigurator : IModuleConfigurator
    {
        /// <summary>
        /// Register Project details with CelloSaaS.
        /// </summary>
        /// <param name="moduleConfiguration">The module configuration.</param>
        public void Configure(ModuleConfig moduleConfiguration)
        {
            //cellohelp:moduleconfiguration
            moduleConfiguration.Add("ProjectModule", "Project Module")

                 // cellohelp:usageconfiguration
                 .WithUsages(u => u.Add("ProjectUsage").WithName("Project
Usage").WithTypeName(UsageTypeConstants.BlockUsage).WithTypeCode(UsageTypeConstants.BlockUsage))

                 // cellohelp:featureconfiguration
                 .WithFeatures(f => f.Add("ProjectFeature").WithName("Project Feature")

                       //cellohelp:entityconfiguration
                       .WithEntities(e => e.Add<Model.Project>())

                             //cellohelp:privilegeconfiguration
                             .WithPrivileges(p => p.Add("Search_Project").WithName("Search Project").WithDescription("user can
search their Project  list")
                                  .Add("View_Project").WithName("View Project").WithDescription("user can View their Project
list")
                                  .Add("Approval_Project").WithName("Approval Project").WithDescription("user can approve their
Project  list")
                                  .Add("Add_Project").WithName("Add Project").WithDescription("user can add their Project
list")
                                  .Add("Edit_Project").WithName("Edit Project").WithDescription("user can edit their Project
list")
                                  .Add("Delete_Project").WithName("Delete Project").WithDescription("user can delete their
Project  list")))
```

Once Config.cs has been updated, it has to be configured in the Web APP>> Global.asax file under the `Configure()`method.

Ex: For the module ProjectManagement for the above ProjectModuleConfigurator, this line is added to the Global.asax

```
CelloSaaS.Configuration.CelloConfigurator.RegisterModule<ProjectManagement.ProjectModuleConfigurato
r>();
```

Note: You cannot have a module partly configured in the XML mode and partly configured via Fluent API.

Parameters for registration of a module using Fluent API:

Method and Parameters	Description
Add(string moduleCode, string moduleName = null)	Specify the Code and Name for the particular Module
WithFeatures(Action<FeatureConfig> featureFunction)	Specify the Feature name
WithName(string moduleName)	Pass the privilege name for the module
WithUsage(Action<UsageConfig> usageFunction)	Pass the usage details for the module
WithService(Action<Service> funService)	Pass the service details for the module
GenerateConfigXml()	To generate the configuration XML file. If we call this method, it create the configuration XML file.
ReadFromXml(string xmlContent)	To configure Module Configuration through XML (.config) file

Final Step

One final step to include the module in the cello web app is

1. Build the cellomodule
2. To the Cello Web app > References add the cellomodule
3. Build the solution and run your application.

4. Contact Information

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

Email: support@techcello.com

Phone: +1(609)503-7163

Skype: techcello