



Cello Framework –DEVELOPER GUIDE

Version 4.2



Table of Contents

1	INTRODUCTION.....	7
1.1	What is CelloSaaS?	7
1.2	How will you build your application?.....	8
1.3	Stakeholders of CelloSaaS	8
1.4	Components of CelloSaaS	9
1.5	Technology Architecture	14
2	REQUIREMENTS	17
2.1	Hardware Requirements	17
2.2	Software Requirements	17
2.3	Installation Guidelines.....	17
3	BUILDING YOUR FIRST HELLO WORLD	19
3.1	Create a Web Application Using CelloSaaS Templates	19
3.2	Configure Your Database	20
3.3	Add View, Service and Data Access Layer	22
4	PLUMBING LAYER	29
4.1	Foundation (Plumbing) Features	29
4.2	Model-View-Controller (MVC).....	30
4.3	Exception handling.....	30
4.3.1	Exception handling Class Members	30
4.3.2	How is Exception handled in CelloSaaS?	31
4.4	Service Post and Preprocessors	32
4.5	Scalable Connection Strings Management	34
4.6	Logging.....	35
4.6.1	Logging Class Members	35
4.6.2	How is logging handled in CelloSaaS?	35
4.7	Scheduler	37
4.7.1	CelloSaaS Event System	37
4.7.2	Time based Scheduler.....	49
4.7.3	Notification Manager [Obsolete]	53
4.7.4	Notification Manager [Current]	55
4.8	System Notifications	72
4.9	Template Management.....	75
4.10	Tenant Encapsulation	84
4.11	Automation of Data Access Layer.....	88
5	SERVICE FACTORIES AND DAL IMPLEMENTATION FACTORIES	98
5.1	Caching	100
5.1.1	AppFabric Cache	100

5.1.2	How to use in Application	107
5.1.3	Advantages of Caching	109
5.1.4	Caching Best Practices	109
6	TENANT CONFIGURATION FRAMEWORK	112
6.1	Data Model Extension	115
6.1.1	Key Pair Value Implementation	115
6.1.2	Configuration Settings	120
6.2	Settings Template	138
6.3	Forms & Grids Customization	142
6.3.1	Form Customization	144
6.3.2	Grid Customization	148
6.4	Pickup List Management	154
6.4.1	Managing Pickup Lists	155
6.4.2	Editing Pickup Lists	156
6.4.3	Deactivate Pickup Lists	157
6.4.4	Pickup List value	158
7	CELLOSAAS SERVICE ENDPOINT MANAGEMENT	164
8	SECURITY FRAMEWORK	170
8.1	Role Management	173
8.1.1	Roles and Access provisions in CelloSaaS	174
8.1.2	Add/Edit Role	174
8.1.3	Delete Role	176
8.2	Service Administrator	176
8.2.1	Assigning Privileges to roles	179
8.3	Privilege Management	184
8.4	Data Access Management	186
8.5	Page Level Access Management	189
8.6	Field Level Access Management	189
8.7	User Entity Access Management	191
8.8	Audit Trail	208
8.9	Triple DES Password Encryption Service	210
8.9.1	Implementation of Triple DES	210
8.10	CelloSaaS - Single Sign On	210
8.11	Admin Data Management	218
9	SUBSCRIPTION	222
9.1	Manage Subscription	222
9.2	Tenant Subscription	232
9.3	Validate Subscription	232



10	TENANT MANAGEMENT	235
10.1	OnBoarding Using Admin Dashboard	235
10.1.1	Add Tenant	235
10.1.2	Edit Tenant.....	237
10.2	One Click Application Subscription	237
10.3	Pre-Requisites for Creating Tenants	239
10.3.1	Post Process.....	240
10.4	Tenant Hierarchy.....	240
10.5	Terminorlogy Used in Tenant Hierarchy:	241
11	METERING	247
12	SETTINGS	254
12.1	System Settings	254
12.1.1	Setting Attribute	254
12.1.2	Tenant Settings.....	256
12.1.3	Role Settings.....	257
12.1.4	User Settings/Personalization	258
12.2	User Authentication Settings.....	260
12.2.1	User Account Locking and Un-Locking	260
12.2.2	Forced Password Change	261
12.2.3	First Time User	262
12.2.4	UserPasswordExpiration [Optional].....	262
12.2.5	PasswordValidationPlugin	263
12.2.6	Hash/Encrypt mechanism for password.....	264
12.2.7	Multiple User and Authentication table.....	264
12.2.8	Membership History	266
12.2.9	User Description	266
12.2.10	Instrument User Login and User Logout.....	266
12.2.11	Lock, Unlock Users	266
12.2.12	Quartz Scheduler	267
12.2.13	User Activity	267
12.2.14	Account Linking and Sharing	268
12.2.15	User Requests the Tenant	271
12.2.16	Tenant Request the User	271
13	DEVELOPER PRODUCTIVITY	275
13.1	Code Generator.....	275
13.1.1	Features	275
13.1.2	Prerequisites	275
13.1.3	Components of code generator.....	276



13.1.4	Steps to follow to start generating the code	283
13.2	Utilities	295
14	WINDOWS COMMUNICATION FOUNDATION (WCF) SERVICE	296
14.1	Secret shared key for invoking services	304
15	CELLOSAAS WORKFLOW	305
15.1	Workflow	305
16	CELLOSAAS BUSINESS RULES ENGINE	338
16.1	What Role Does CelloSaaS BRE Play in Your Business?	338
16.2	Crafting a Business Rule	339
16.3	Components of Business Rules	340
16.4	Rules Library	340
16.5	Rules Web Service Application	340
16.6	Silverlight Client Application	340
16.6.1	Why Silverlight?	340
17	QUERY BUILDER	358
17.1	Chart builder	361
17.2	Dynamic variables in query builder	364
17.3	Report builder	365
17.4	Product Analytics	372
18	OBJECT-RELATIONAL MAPPING	376
18.1	Entity Framework	376
18.1.1	Introduction	376
18.1.2	Why do we need Entity framework?	376
18.1.3	Purpose of Entity framework	376
18.1.4	Entity Framework – Key Features	376
18.1.5	Entity Framework – Approach	377
18.1.6	Entity framework – Code First	377
18.1.7	Entity framework – Model First	377
18.1.8	CelloSaaS integration with Entity Framework:	378
18.1.9	Extended field support in Entity Framework	378
18.1.10	Implementing Entity Framework in CelloSaaS	378
19	HOSTING YOUR SAAS APPLICATION	389
19.1	Deploying CelloSaaS in Windows Azure	389
19.2	Scalability in CelloSaaS	390
19.3	App Fabric Installation	391
20	CELLOSAAS CACHE SYNCHRONIZATION	402
21	UTILITIES	403
22	APPENDIX	405



23 CONTACT INFORMATION	407
------------------------------	-----

1 INTRODUCTION

Software as a Service (SaaS) is a paradigm shift in the way software is developed, purchased and used. “Pay only for what you use” mantra is widely accepted worldwide and companies like SalesForce.com, SuccessFactors.com, etc. thrive on this principle. Customers (end users) stand to gain a lot of benefit – no upfront investment - CAPEX, better management of cash flow as OPEX, efficient management of users/utilization of the application, etc. It is being increasingly adopted by both software buyers and software makers.

SaaS (also known as On-demand solutions or hosted software solution) is a software application delivery model where an ISV develops software and hosts/operates over the Internet. Software users pay the vendors on a “pay as you go” basis, usually a monthly or yearly subscription fee. In this model, the software users can access the software from anywhere over the Internet.

This hosted software solution is adopted in line of business services like CRM, HR, Payroll processing, etc. Consumer related services like Google Docs, Web-based-mails, and online banking are also on-demand solutions. SaaS has gained acceptance not only among the non-critical business processes like above, but some of the software products that service critical business functions (e.g. Hospital management systems) are also considering SaaS more actively for some of their product lines. Popularity of Cloud Computing principles such as IaaS and PaaS are also accelerating the adoption of SaaS.

What makes SaaS more desirable?

Both the software users and software vendors would like to be associated with on-demand solutions and there are several good reasons behind this trend. Some key attractions causing this increased adoption are highlighted below.

For Software vendors:

Single instance of the product services all the customers

Upgrade and maintenance are one step

Deployment process is simple and consumes less time and cost

Customer acquisition is faster and easier; Shorter sales cycle

Recurring predictable monthly revenue

Access to the long tail

For Software buyers:

Zero CapEx and pay as you go pricing structure helps manage cash flow better

Add users as you go

No additional hardware needed

No initial installation and implementation struggle

Less or no need for internal IT resources

Easy-to-use, mostly very intuitive and less training requirements

1.1 What is CelloSaaS?

Given the benefits and attractions in SaaS, many traditional software providers have started giving a place to SaaS in their product roadmap. However, moving a traditional software product to SaaS is not as simple as hosting the existing on-premise product in a data center and making it available for



organizations over the Internet. Huge efforts are involved in reengineering and remodeling the product to deliver it as a SaaS solution.

Building a new SaaS solution or moving an existing on-premise product to a SaaS solution involve building scalable architecture for the product to support multiple customers. This foundation building process typically takes about 4 to 6 months before the core business functionalities of the solution are built. So, anything to reduce this development time can be of significant help to a Software vendor, thus speeding up the time to market.

TechCello's CelloSaaS, a powerful enterprise multi-tenant product development framework is a comprehensive platform, for building web-based, on-demand, software-as-a-service (SaaS) applications. It is a development framework that has the basic plumbing layers required for building any software products / applications such as caching, logging, alerts, etc. and SaaS building blocks such as multi-tenancy, tenant provisioning, configurability, license management, metering, etc.

The Framework makes life easier for Application Developers, ISV's / Service Providers & Business Leaders. You can create sophisticated multi-tenant, SaaS applications rapidly without the high upfront costs. This framework can reduce about 30-40% of the development time for building a multi-tenant application.

1.2 How will you build your application?

This guide targets software architects and developers building applications with CelloSaaS. As a developer, you are already familiar with most standard development tools and have experience with building web applications. This manual will help you to get started with building your on-demand, SaaS application using CelloSaaS framework.

1.3 Stakeholders of CelloSaaS

To understand the rationale for the technical design of the CelloSaaS framework, it is beneficial to understand some of the major stakeholders and user roles in SaaS business model. While there are many players, fundamentally there are two major stakeholders and two major user roles in a web-delivered SaaS framework. The two major stakeholder roles are:

SaaS provider: Owns the SaaS framework and provides different services.

Infrastructure services include security, performance monitoring.

Application services (e.g. Payroll processing, procurement platform, etc.)

Tenant services include billing, service level agreements, and contracts.

Tenants: Utilize one or more applications in the SaaS platform. This stakeholder consumes the SaaS application. The two major user roles are:

Developers who use the services of the platform provider to develop, test, and deploy new applications or new releases of the application. Developers need to understand the data model that supports multi-tenancy before designing their application.

End users who use the features of one or more applications.

1.4 Components of CelloSaaS

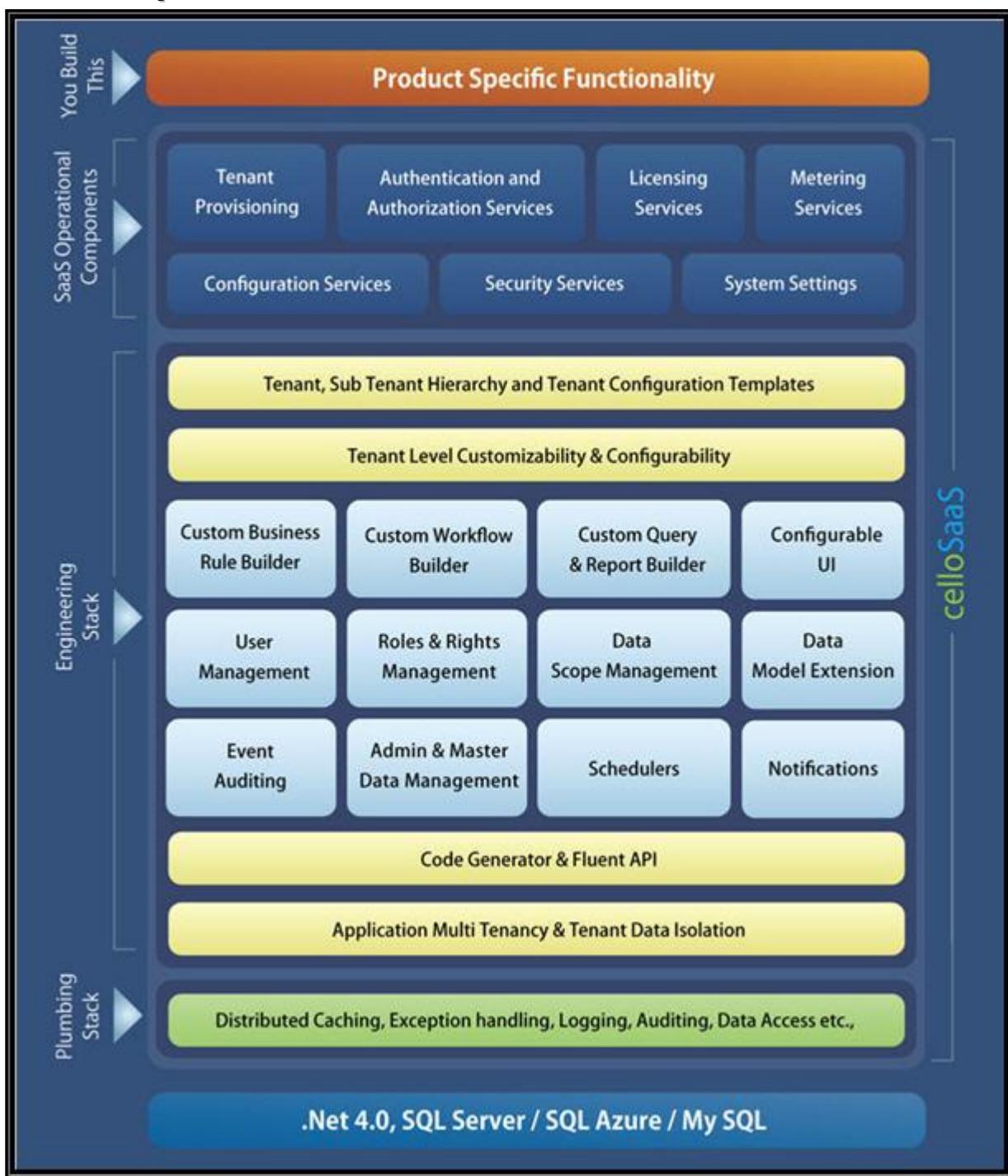


Figure 1-1 – Components of CelloSaaS

Basic Plumbing	
Performance Instrumentation	<p>Install, Manipulate, Capture, log</p> <ul style="list-style-type: none"> Performance data can be captured for any or all methods
Post and Pre Processors	<p>Hook pre and post processing logic for CRUD operations</p> <ul style="list-style-type: none"> Support the unique needs of each application
Notifications	<p>Custom Notification through emails and FTP</p> <ul style="list-style-type: none"> Tenant can configure their email notification text & templates <p>Map placeholder entities</p>
MVC architecture	Provides high level of modularity
Service factories & DAL Implementation Factories	<p>Plug-in using configuration</p> <ul style="list-style-type: none"> Different Service Implementation and DAL implementation can be easily plugged in using configuration
Scheduler	Schedule various activities based on time and events
Caching	<p>Methods for Add, Remove, etc</p> <ul style="list-style-type: none"> Set up default cache manager in config file
Exception Handling	<p>Configure and extend handlers</p> <p>Rethrow exceptions</p> <p>Prioritize exceptions</p> <p>Write exceptions to event log</p> <p>Write exceptions to database</p>
Data Access Layer (DAL)	<p>Designed to support multiple databases</p> <ul style="list-style-type: none"> Designed to work with multiple databases – MS SQLServer, SQL Azure, MySQL, etc. by just changing the provider in config file. Data Provider currently available for MSSQL 2008, SQL Azure and MYSQL. Other Data providers can be custom developed on request. <p>Tenant Isolation & DAL Automation</p>
Data Isolation and Scale-out Schemes	<p>Separate DB per tenant</p> <p>Separate table per tenant</p> <p>DB or Table per application</p> <p>Shared DB and tables</p>

SaaS Customization Framework

Tenant based URL Mapping	Each tenant can be mapped to a different URL
Data Model Extension	Add more columns to existing data model entities
Skin Configuration	Customize the Themes, Logos, etc. for each tenant
Forms and Grids Customization	Customize the grid and form labels Manage the visibility of the form and grid columns Manage the editability of the form columns
Pickup List Management	Manage pickuplists Manage pickup list values Manage the relationship between pickup lists
Business Rule Configuration	Add custom business rules through XML End user customizable (UI based) Business Rules template Business Rule Chaining Tenant aware and isolated business rules execution
Report Customization	Customize the labels of tabular reports Manage visibility of the report columns Custom Query Builder and Report builder
Work Flows Management	End user Customizable Workflows Drag and Drop Workflow customization tool Developer control over Workflow configuration and execution Parallel execution of multiple workflows Cascading configuration from Parent to Tenant to Sub tenant Automated and Manual Workflow Configuration Governance Dashboard
Tenant Configuration Template	Group configurable parameters into a template and assign it to Tenants. Roll down Parent Tenant configurations to sub tenants. Allow sub tenants and users to change configurable attributes

SaaS Security Framework

Role Management

Manage global roles and roles specific to each tenant

Privilege Management	Manage the privileges in the product and restrict access based on privileges
Role Access Management	Map privileges to roles
Data Level Access Management (Data Scope policies)	<p>Restrict access at a data level based on privileges</p> <ul style="list-style-type: none"> Example: Manager can see only his subordinate details or department details. Sales person can see only data related to his territory.
Page Level Access Management	Restrict access to menus and pages based on privileges
Field Level Access Management	<p>Restrict field level access based on privileges</p> <ul style="list-style-type: none"> Example: only HR can see the salary details
Audit Trial	<p>Audit log of events and transactions</p> <p>Track CRUD changes at object level</p> <p>Track field level changes to objects</p> <ul style="list-style-type: none"> Track and analyze changes such as Create, Read, Update, Delete.

SaaS Administration	
Tenant Data Management	<p>Create multi tenant screens and simple data entry screens through XML configuration</p> <ul style="list-style-type: none"> Example : Master screens can be created in 15 min by adding configuration in master config file
Tenant Hierarchy Management	<p>Administer group of tenants by a Service Manager, Tenant Scope Management</p> <p>Manage Relationship between users.</p> <p>Multi level Tenant - Sub tenant - user management</p> <p>Roles and privilege management for nth level user</p>
Fluent API	Register application related configuration information into the Techcello framework using FluentAPIs

SaaS Package Management	
Manage / Validate Packages	<p>Create Packages by enabling and disabling various sub-systems modules and features of a product / product suite / product line</p> <p>Assign these packages to specific Tenants</p> <p>Users within each tenant can use only those features, modules and menus enabled for their tenant.</p>

SaaS Tenant Management

Manage Tenants	Provision new tenants, update and delete
Customize / Extend attributes	Configure Custom Attributes and extend based upon their specific needs
Manage Packages for each Tenant	Create and manage group of features into packages

SaaS User Management

Manage User	Create, update and delete users
Customize user attributes	Register application related configuration information into the Techcello framework using FluentAPIs
Manage User Roles	Map users to global as well as tenant specific roles
Share Users across Tenants	Share users across multiple tenants upon approval from the user's own Tenant Ready to use screens for User Approval and Disapproval

SaaS Metering

Metering / Usage recording	Record / Meter the usage of any entity, transaction or event
Usage Reports	View the usage details – user wise and tenant wise

SaaS System Settings

Tenant Settings	Configure System settings at a tenant level
Role Settings	Configure System settings at a role level within each Tenant
User Settings	Configure System settings at a user level

Development Productivity

Code Generator (Integrated with Visual Studio)	Creates basic scaffolding as required by the Techcello framework Generates loosely coupled and well structured code Complete control over the generated source Code Change and configure the Code generator plug-in
--	--

1.5 Technology Architecture

CelloSaaS is built on top of the following technology stacks:

- .NET 4.0, C#
- ASP.NET MVC 3.0
- SQLServer 2008

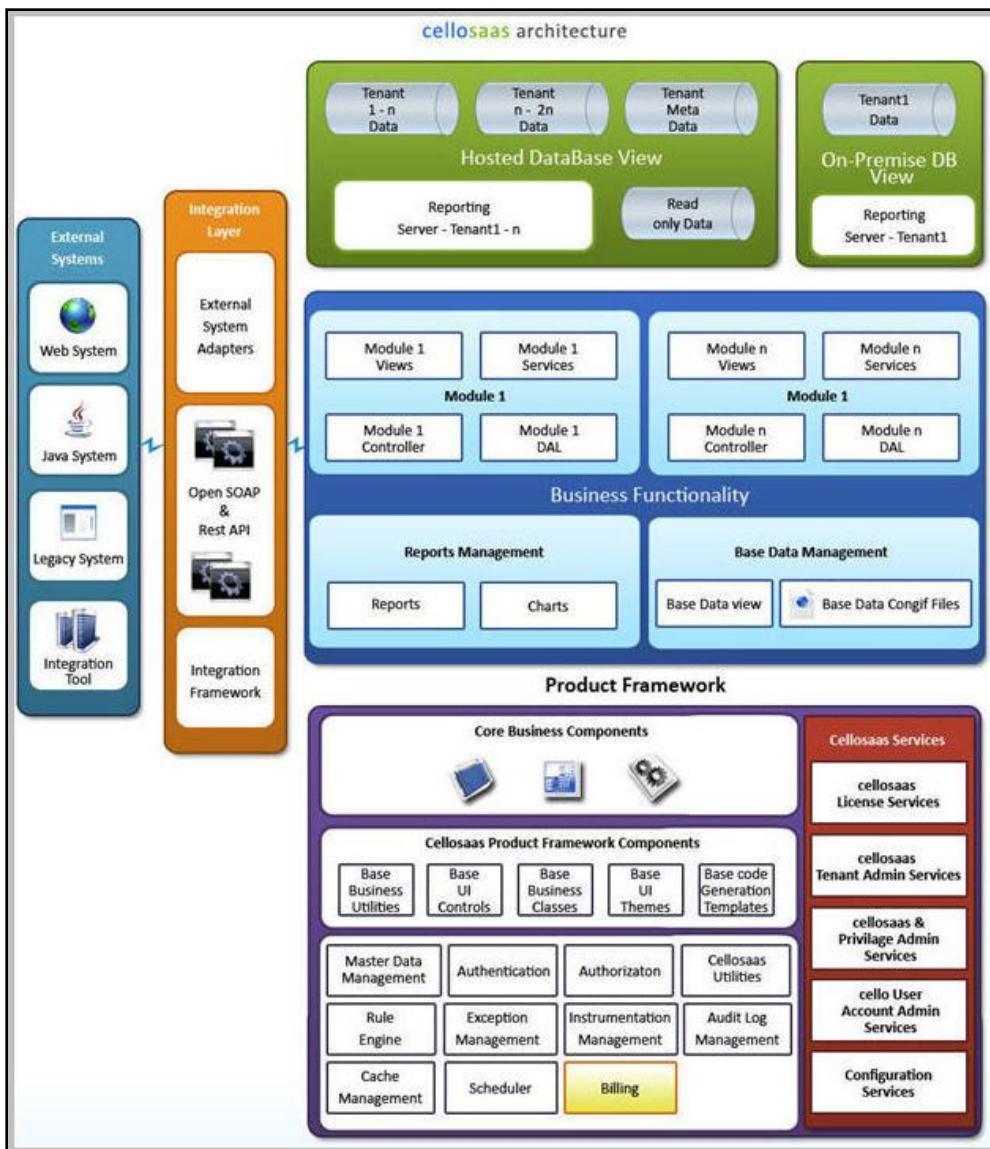


Figure 1-2 – CelloSaaS Architecture

CelloSaaS is built in a way such that it is easily extendable by the developers. It follows a complete layered design; and both the layers and modules are loosely coupled. The above diagram depicts the typical architecture of a product built using CelloSaaS. You can notice that the entire product is built on top of CelloSaaS components and Services.

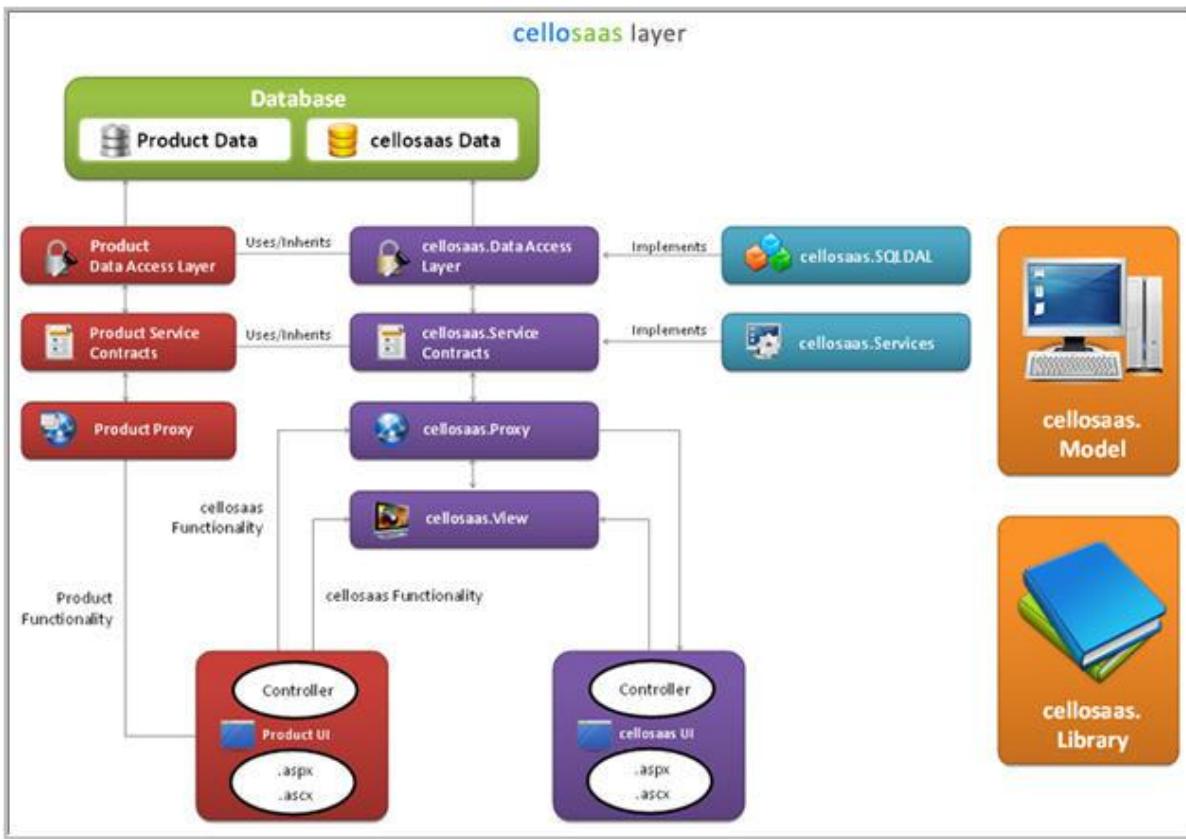


Figure 1-3 –CelloSaaS Layer Diagram

CelloSaaS is composed of the following assemblies.

Table 1.1 – CelloSaaS Assemblies

Assembly Name	Description
cellosaas.Library	This is the core assembly which has all the base functionalities required by the rest of the layers
cellosaas.Mode	This assembly contains all the entities of CelloSaaS
cellosaas.DataAccessLayer	This assembly contains the interfaces for all the data access components
cellosaas.SQLDAL	This assembly contains the SQL implementation of the data access layer

cellosaas.ServiceContracts	This assembly contains the interfaces of all the services
cellosaas.Services	This assembly contains all the CelloSaaS services
cellosaas.Proxies	This assembly acts as a facade layer for the services
cellosaas.View	This assembly contains the view level components
cellosaas.UI	This contains the controller and .aspx, .ascx files for the administration module

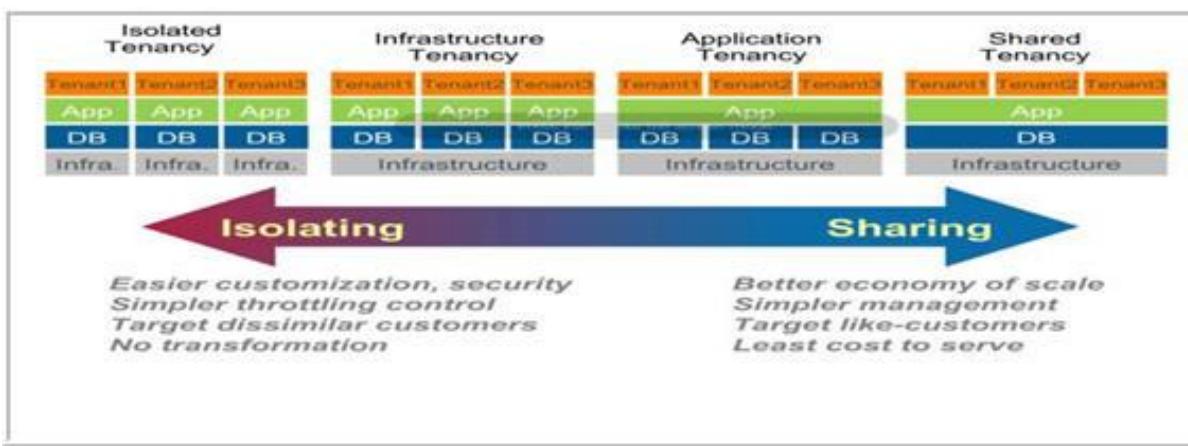


Figure 1-4 – Data Architecture

CelloSaaS supports and follows “**Shared Schema, Shared table**” mode of multi tenant data architecture. However, it is also possible to support the SaaS product with the below database models

Shared Database

Isolated Schema

Isolated database

CelloSaaS also supports a mixed mode where the database is partitioned horizontally by tenants for better scalability.

2 REQUIREMENTS

CelloSaaS leverages Microsoft's enterprise library and ASP. NET MVC. This section provides a list of resources, which will help you get started with CelloSaaS.

2.1 Hardware Requirements

A 3 gigahertz Pentium processor.

At least 2 GB of RAM (Recommended 4 GB RAM).

2.2 Software Requirements

Development: Windows Server 2003 / 2008, Windows 7, Windows Vista

Deployment: Windows Server 2003 / 2008

Microsoft .NET Framework 4.0

ASP.NET MVC 3.0

Microsoft Visual Studio 2010 Professional/Express Edition

Enterprise Library for .NET Framework 4.0

Note: Though Cello framework is compatible with any Operating system, the Installer provided with Cello Package will not support Windows XP, so users who are still using Windows XP can contact Cello team to get information about the installation procedure.



CelloSaaS is yet to support Enterprise Library 5.0. Please do not install Enterprise Library 5.0 on your system for using CelloSaaS.

Visual Studio 2010 Software Development Kit (SDK).

Guidance Automation Extension (GAX) 2010.

Guidance Automation Toolkit (GAT) 2010.

Note: Cello Utilizes Microsoft Distributed Transaction CoOrdinator for Transactions. This needs to be enabled on the developer machine. Refer this [link](#) to know more about MSDTC.

2.3 Installation Guidelines

After purchasing the appropriate License from CelloSaaS, you will receive an email containing the information to download the product from the Techcello website. Once you have the product in place, follow the below procedure to get started with CelloSaaS

Step 1: Unzip the package. It contains the following folders.

- DDL Scripts
- DML Scripts
- DLLs
- Installer



Step 2: Follow the below steps to Install CelloSaaS Framework

- Install Microsoft Guidance Package.
- Install CelloSaaSPackage.vsix

Step 3: Database Creation

Typically, the application built with CelloSaaS uses two schemas:

- CelloSaaS Schema – To store CelloSaaS Metadata
- Application Schema – To store the application data
-

Step 4: Create Metadata Database

- In Object Explorer, connect to an instance of the SQL Server Database Engine and then expand that instance.
- Right-Click Databases and then click New Database.
- In New Database, enter a Database name.
- To create the database by accepting all default values, click OK; otherwise, continue with the following optional steps.
- Execute the DDL scripts to create all the CelloSaaS meta-data tables. These tables are responsible for preserving the metadata for all the components such as Membership, Tenant management, Package Management etc.
- Execute the DML scripts to Insert default values into the metadata tables.

Step 5: Create Application Database

- Follow the step no 4 and create a database for application data.

1. CelloSaaS supports only MS SQL Server 2. After executing the above scripts, make sure there are no errors while execution. 3. Once you complete the above process, refresh the tables list and make sure all the objects are created successfully with the default data.



If anyone of the table is missed or if there is a failure while executing DML script, then the application might not work as expected.

3 BUILDING YOUR FIRST HELLO WORLD

CelloSaaS application framework is a software framework that is designed to support the development of multi tenant or single tenant web applications, dynamic websites, and web services. The framework typically provides core functionality common to most web applications.

3.1 Create a Web Application Using CelloSaaS Templates

Open Visual Studio 2010.

Create a new Project by clicking on Create→ New→ Project button in the home Page.

Under Installed templates section, select Guidance Package and choose CelloSaaS Package.

Name your Solution folder and Click Ok button.

Name your Project or leave the name as same as the Solution Name.

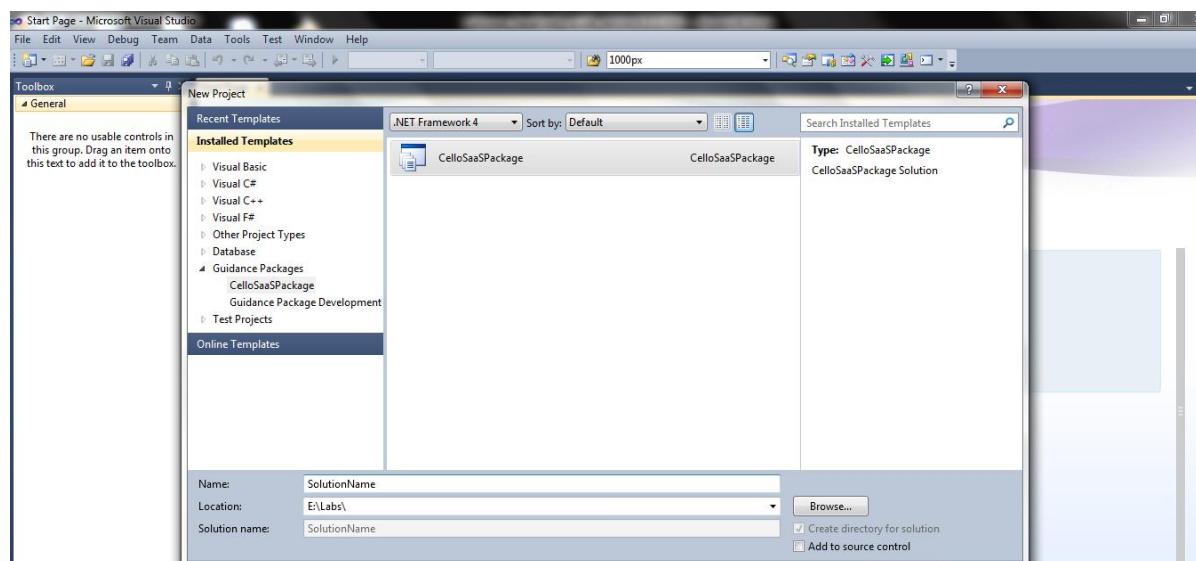


Figure 3-1 – New Project Screen

CelloSaaS creates an empty Solution with the below layers. They are

DAL [Data Acces Layer]

Services

BL[Business Layer]

Web Layer

For the Web tier, CelloSaaS creates a MVC3 web application with default capabilites and administration screens.

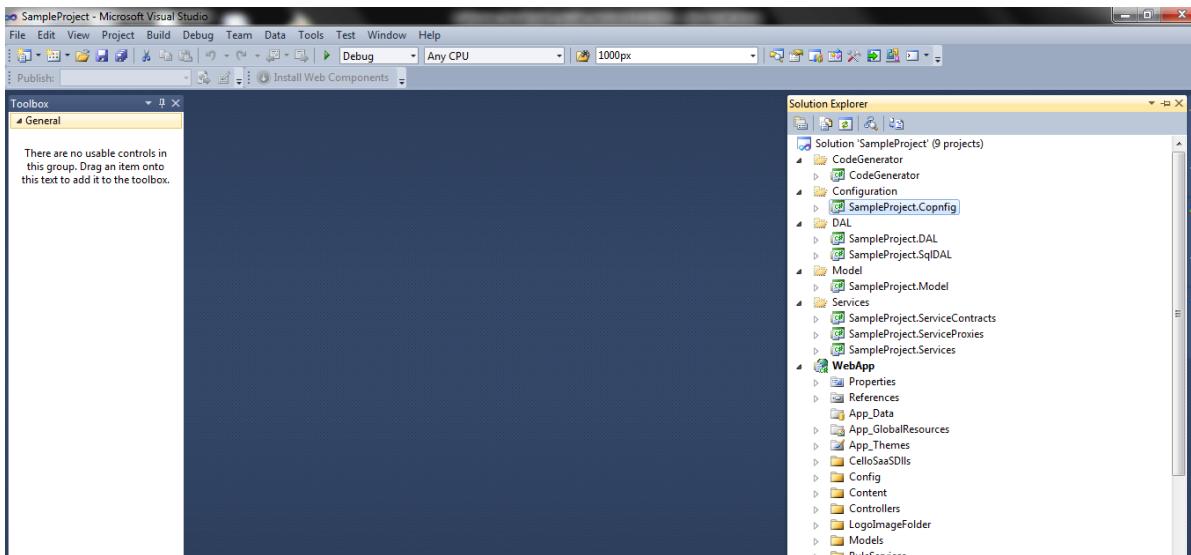


Figure 3-2 – Solution Explorer Screen

3.2 Configure Your Database

In CelloSaaS, there are basically two kinds of data, 1.Metadata 2. Application Data. You can choose to use a separate database for holding CelloSaaS Metadata and another for storing application data, or maintain a single shared database for both. It's the decision that has to be made before starting the development. CelloSaaS recommends the developer to use two different databases, as it is easy to maintain and upgrade CelloSaaS versions in the future and to balance the load.

Open SQL. Config file present at the root of the web application and configure the database configurations (update database connection strings – pointing to the server name, authorization credentials, etc.).

ApplicationConnectionString - Database information for your product/Application
CelloSaaSConnectionString - Database information for CelloSaaS
UserConnectionString - Database information for Membership Details
WorkFlowConnectionString - Database information for WorkflowMetadata Details
ProductAnalyticsConnectionString - Database information for storing Product Analytics details

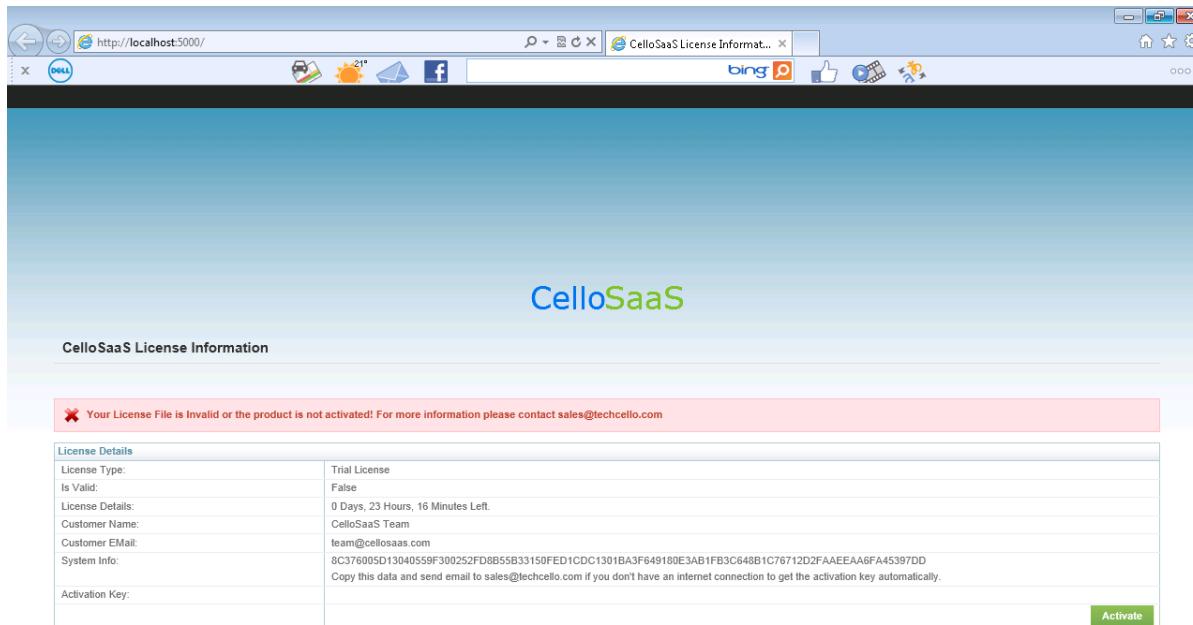
The sample configuration mentioned below specifies that both CelloSaaS and your application use the same database. In addition to these connection strings, you can define more for the application.

```
<connectionStrings>
<add name="ApplicationConnectionString" connectionString="DataSource=ServerName;Initial Catalog=DatabaseName;User Id=Database UserName;Password=DatabasePassword;" providerName="System.Data.SqlClient"/>
<add name="CelloSaaSConnectionString" connectionString="Data Source=ServerName;Initial Catalog=DatabaseName;User Id=Database UserName;Password=DatabasePassword;" providerName="System.Data.SqlClient"/>
<add name="UserConnectionString" connectionString="Data Source=ServerName;Initial Catalog=DatabaseName;User Id=Database UserName;Password=DatabasePassword;" providerName="System.Data.SqlClient"/>
<add name="WorkFlowConnectionString" connectionString="Data Source=ServerName;Initial Catalog=DatabaseName;User Id=Database UserName;Password=DatabasePassword;" providerName="System.Data.SqlClient"/>
```

```
<add name="ProductAnalyticsConnectionString" connectionString="Data
Source=ServerName;Initial Catalog=DatabaseName;User Id=Database
UserName;Password=DatabasePassword;" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Build and run the application;

You may get this screen initially when you build and run the application for the first time. This is due to invalid/no license file placed in the project. As you may know Cello Framework requires a valid license file in order to work properly.



Follow the below steps to place the license file in the project:

- 1. Navigate to bin folder in the Web Project
- 2. Create a folder inside the bin folder named "License"
- 3. Drop the License file inside the folder and run the solution again [License File shared with you via Email]

[Note: Please make sure no space in the license file path for Ex:

Correct: E:\Labs\LMS\WebApplication\bin\License\CelloSaaS.License.lic
InCorrect: E:\Labs\LMS\Web Application\bin\License\CelloSaaS.License.lic]

Once you place the license file in the appropriate folder, you can then go ahead and run the application.

For more information on License Activation. Refer the guide [here](#)



Figure 3-3 – CelloSaaS login Screen

3.3 Add View, Service and Data Access Layer

CelloSaaS is developed using n-tier architecture, in which SaaS framework UI, business logic code and data access code are placed in separate layers namely Presentation layer, Business logic layer and Data access layer respectively. The interface that is used between a Presentation layer and Business logic layers is Controller, which prepares Business objects and passes to Business logic layer. The business logic layer in turn calls the appropriate Data access layer method. The data access layer uses SQLDAL as an interface in order to communicate with the database.

Once the user request is processed as a query, the SQLDAL will return the query result to data access layer. Data access layer will prepare the Business object based on the resultant query and returns the object to Business logic layer. The Business object layer returns the object received from Data access layer to the called method of the presentation layer. Finally, in the presentation layer, the received business object values will be assigned to the corresponding UI controls of the page, and this will be displayed to the user.

Ideally, the SaaS product should be organized in a structure similar to CelloSaaS.

Model: This should contain the business entities.

ServiceContracts: This should contain interfaces for the business services.

Services: This should contain the business service implementations.

DALLayer: This should contain the interfaces for the data access classes.

SQLDAL: This should contain the SQL implementation for the data access classes.

In addition to this, the classes you add to these layers should follow specific design norms, which are described below:

The classes you add to these layers should follow specific design norms such as Model, EntityIdentifier, ExtendedRow and EntityDescriptor. These properties are used to obtain extension features given by CelloSaaS.

- **Model:** You should have your business entities in Model Layer. You will have to add CelloSaaS model reference (CelloSaaS.Model) in this project. Each model has to be inherited from “CelloSaaS.Model BaseEntity” which is provided by CelloSaaS model.
- **ExtendedRow** is used to have extended field details. Extended column values will be stored in the form of either key-value pair or Flat table model [Refer Chapter Data Model Extension].
- **DAL:** To support multiple databases such as Microsoft SQL Sever, MS SQL Azure, AWS RDS - SQL Server etc, CelloSaaS suggests having a data access layer with the interface. You will have to create a project with the Interfaces of the Data access classes. These classes should be suffixed with the name “DAL”. Assembly name of this project should be with suffix “DAL”.

Example: EmployeeManagement.DAL

Every interface must be inherited from CelloSaaS.DAL IEntity<T> which is provided by CelloSaaS. Structure of CelloSaaS.DAL.IEntityDAL<type> is as follows:

```
Public interface IEntityDAL<T>
{
    ///<summary>
    /// Used to call DoCreate and RaiseCreatedEvent
    ///</summary>
    string Create (DataCreateRequest dataCreateRequest);
    ///<summary>
    /// Used to call RaiseFetchingEvent, DoFetch and RaiseFetchedEvent
    ///</summary>
    T Fetch (DataFetchRequest dataFetchRequest);
    ///<summary>
    /// Used to call RaiseSearchingEvent, DoSearch and RaiseSearchedEvent
    ///</summary>
    Dictionary<string, T> Search (DataSearchRequest dataSearchRequest);
    ///<summary>
    /// Used to call DoUpdate and RaiseUpdatedEvent
    ///</summary>
    void Update (DataUpdateRequest dataUpdateRequest);
    ///<summary>
    /// Used to call RaiseDeletingEvent, DoDelete and RaiseDeletedEvent
    ///</summary>
    void Delete (DataDeleteRequest dataDeleteRequest);
    ///<summary>
    /// Used to call DoSoftDelete
    ///</summary>
    void SoftDelete(DataDeleteRequest dataDeleteRequest);
}
public interface IEmployeeDetailsDAL : IEntityDAL<EmployeeDetails>
{ }
```

```
Example
public interface IEmployeeDetailsDAL : IEntityDAL<EmployeeDetails>
{ }
```

- **SQL:** The SQL implementation of the data access classes will be in a separate assembly. In CelloSaaS, we have to follow a naming convention for the assembly name. Assembly name should be with <Provider name (Database provider)>DAL.

Example: EmployeeManagement.SqlDAL

Every implement class must be inherited EntityDAL<T> which is provided by CelloSaaS. In this class, you will have to override the following operations.

```

///<summary>
/// Builds the insert query and executes.
///</summary>
///<param name="createRequest">Which has entity object</param>
///<returns>CreatedIdentifier</returns>
string DoCreate(DataCreateRequest dataCreateRequest)
///<summary>
/// Builds the fetch query,executes and returns the fetched object
///</summary>
///<param name="dataFetchRequest">Which has Identifier,Datascope and
EntityBaseDatascope</param>
///<returns>Fetched object</returns>
TDoFetch(DataFetchRequest dataFetchRequest)
///<summary>
/// Builds the search query,executes and returns the fetched results as key-value collections.
///</summary>
///<param name="dataSearchRequest">Which has list of
Identifier,EntityIdentifier,searchCondition,Datascope and EntityBaseDatascope</param>
///<returns>Key-Value collection of fetched object</returns>
Dictionary<string, T> DoSearch(DataSearchRequest dataSearchRequest)
///<summary>
/// Builds the update query and executes.
///</summary>
///<param name="dataUpdateRequest">Which has Identifier,Entity object</param>
///<returns></returns>
void DoUpdate(DataUpdateRequest dataUpdateRequest)
///<summary>
/// Builds the delete query and executes.
///</summary>
///<param name="dataDeleteRequest">Which has Identifier,Entity</param>
void DoDelete(DataDeleteRequest dataDeleteRequest)
///<summary>
/// Builds the update query for softdelete and executes.
///</summary>
///<param name="dataDeleteRequest">Which has Identifier,Entity</param>
void DoSoftDelete(DataDeleteRequest dataDeleteRequest)

```

Example: EmployeeDetailsDAL

```

protectedoverridestring DoCreate(DataCreateRequest createRequest)
protectedoverridewithoutreturn void DoDelete(DataDeleteRequest dataDeleteRequest)
protectedoverrideEmployeeDetails DoFetch(DataFetchRequest dataFetchRequest)
protectedoverrideDictionary<string, EmployeeDetails> DoSearch(DataSearchRequest
dataSearchRequest)
protectedoverridevoid DoUpdate(DataUpdateRequest dataUpdateRequest)
protectedoverridevoid DoSoftDelete(DataDeleteRequest dataDeleteRequest)

```

In every operation, you will have to pass the respective DataRequest which has the following properties.

DataFetchRequest

```
///<summary>
/// Primary Id of the record
///</summary>
string Identifier
///<summary>
/// Datascope for demanded privilege
///</summary>
DataScope DataScope
///<summary>
/// EntityBasedDatascope of the entity for demanded privilege
///</summary>
EntityBasedDataScope EntityBasedDataScope
```

DataSearchRequest

```
///<summary>
/// List of primary Ids which are to be fetched
///</summary>
string[] Identifiers
///<summary>
/// Entity Identifier of the business object
///</summary>
string EntityIdentifier
///<summary>
/// Search condition if required any
///</summary>
ISearchCondition SearchCondition
///<summary>
/// Datascope for demanded privilege
///</summary>
DataScope DataScope
///<summary>
/// EntityBasedDatascope of the entity for demanded privilege
///</summary>
EntityBasedDataScope EntityBasedDataScope
```

DataUpdateRequest

```
///<summary>
/// Primary Id of the record
///</summary>
string Identifier
///<summary>
/// Entity object which is going to be updated
///</summary>
 BaseEntity Entity
```

DataDeleteRequest

```

///<summary>
/// Primary Id of the record
///</summary>
string Identifier
///<summary>
/// Entity object which is going to be deleted
///</summary>
 BaseEntity Entity

```

- **Services:** Every service has to be inherited from CelloSaaS.Services.BaseService which is provided by CelloSaaS.

```

•
///<summary>
/// To validate access for given entity and entityAction
///</summary>
///<param name="entityAction">EntityAction will have EntityOperation and PrivilegeContext</param>
///<param name="entity">Business object</param>
///<returns></returns>
bool ValidateAccess(CelloSaaS.Model.EntityAction entityAction, BaseEntity entity)
///<summary>
/// To validate access for list of entity with given EntityAction
///</summary>
///<param name="entityIdentifier">EntityIdentifier of business object</param>
///<param name="entityAction">EntityAction will have EntityOperation and PrivilegeContext</param>
///<param name="entityList">Key value collection of entities.Key-Identifier,Value-Entity</param>
///<returns>Key value collection of result. Key-Identifier, Value- Access result</returns>
Dictionary<string, bool> ValidateAccess(string entityIdentifier, CelloSaaS.Model.EntityAction
entityAction, Dictionary<string, BaseEntity> entityList)
///<summary>
/// To validate access for given entity with permissionlist
///</summary>
///<param name="permissionlist">List of permissions</param>
///<param name="entity">Business Entity</param>
///<returns>Key value collection of result. Key-permission, Value-Access result</returns>
Dictionary<string, bool> ValidateAccess(string[] permissionlist, BaseEntity entity)
///<summary>
/// To validate access for given referenceId,entityIdentifier with permissionlist
///</summary>
///<param name="permissionlist">List of permission</param>
///<param name="entityIdentifier">EntityIdentifier of business object</param>
///<param name="referenceId">Referenceld for which access needs to be checked</param>
///<returns>Key value collection of result.Key-Permission, Value-Access result</returns>
Dictionary<string, bool> ValidateAccess(string[] permissionlist, string entityIdentifier, string
referenceId)
///<summary>
/// To set dataScope and entityBasedDataScope in out parameter for given entityIdentifier and
permission.
///</summary>
///<param name="entityIdentifier"> EntityIdentifier of business object </param>
///<param name="permissionName">Privilege name</param>
///<param name="dataScope">Datascope as out parameter</param>
///<param name="entityDataScope">EntityDatascope as out parameter</param>
void SetDataScopes(string entityIdentifier, string permissionName, outDataScope dataScope,
outEntityBasedDataScope entityDataScope)
///<summary>

```

```

/// To set list dataScope and entityBasedDataScope in out parameter for given entityIdentifier and
list of permission.
///</summary>
///<param name="entityIdentifier"> EntityIdentifier of business object </param>
///<param name="permissionlist">List of permission</param>
///<param name="dicDataScope">Key value collection of datascope.Key-Permission,Value-
Datascope</param>
///<param name="dicEntityDataScope">Key value collection of entityDatascope. Key-
Permission,Value-EntityDatascope</param>
public void SetDataScopes(string entityIdentifier, string[] permissionlist, out Dictionary<string,
DataScope> dicDataScope, out Dictionary<string, EntityBasedDataScope> dicEntityDataScope)

```

BaseService allows raising the following events while doing CRUD operation. As in the DAL, these events will not be called by CelloSaaS. The events need to be called whenever required.

```

///<summary>
///<param name="args"></param>
void RaiseFetchedEvent(FetchEventArgs args)
///<summary>
/// This event can be raised before calling DoFetch method of DAL
///</summary>
///<param name="args"></param>
void RaiseFetchingEvent(FetchEventArgs args)
This event can be raised after calling DoFetch method of DAL
///<summary>
///<
///<summary>
/// This event can be raised after calling DoCreate method of DAL
///</summary>
///<param name="args"></param>
void RaiseCreatedEvent(CreateEventArgs args)
///<summary>
/// This event can be raised before calling DoCreate method of DAL
///</summary>
///<param name="args"></param>
void RaiseCreatingEvent(CreateEventArgs args)
///<summary>
/// This event can be raised before calling DoUpdate method of DAL
///</summary>
///<param name="args"></param>
void RaiseUpdatingEvent(UpdateEventArgs args)
///<summary>
/// This event can be raised after calling DoUpdate method of DAL
///</summary>
///<param name="args"></param>
void RaiseUpdatedEvent(UpdateEventArgs args)
///<summary>
/// This event can be raised before calling DoDelete method of DAL
///</summary>
///<param name="args"></param>
void RaiseDeletingEvent(DeleteEventArgs args)
///<summary>
/// This event can be raised after calling DoDelete method of DAL
///</summary>
///<param name="args"></param>
void RaiseDeletedEvent(DeleteEventArgs args)

```

- **View:** Every view page which you create in your application has to inherit from “CelloSaaS.View.celloViewPage” provided by CelloSaaS.

```
<%@Page Title="" Language="C#" MasterPageFile="~/Views/Shared/MasterPage.Master"
Inherits="CelloSaaS.View.celloViewPage"%>
```

The base page does the common activities such as setting the right theme.

- **Controller:** Every controller which is created, has to be inherited from “CelloSaaS.View.celloController” [Mandatory] . CelloController performs the following things. They are
 - Handles unknown actions, feature utilization, unhandled exception, and logs the entire details.
 - Authenticate each AJAX actions and expire the session if not authenticated.

Example

```
Public class EmployeeController : CelloSaaS.View.CelloController
{
}
```

4 PLUMBING LAYER

Microsoft Enterprise Library consists of reusable software components that are designed to assist developers with common application development challenges. It includes a collection of functional application blocks addressing specific cross-cutting concerns such as data access, logging, or validation; and wiring blocks, Unity and the Interception/Policy Injection Application Block, designed to help implement more loosely coupled testable, and maintainable software systems.

Different applications have different requirements, and you will find that not every application block is useful in every application that you build. Before using an application block, you should have a good understanding of your application requirements and of the scenarios that the applications block is designed to address.

CelloSaaS framework uses the following application building blocks from Enterprise Library:

Caching Application Block: Developers can use this application block to incorporate a cache in their applications. Pluggable cache providers and persistent backing stores are supported.

Cryptography Application Block: Developers can use this application block to incorporate hashing and symmetric encryption in their applications.

Data Access Application Block: Developers can use this application block to incorporate standard database functionality in their applications, including both synchronous and asynchronous data access and returning data in a range of formats.

Exception Handling Application Block: Developers and policy makers can use this application block to create a consistent strategy for processing exceptions that occur throughout the architectural layers of enterprise applications.

Logging Application Block: Developers can use this application block to include logging functionality for a wide range of logging targets in their applications. This release further improves logging performance.

Policy Injection Application Block: Powered by the Interception mechanism built in Unity, this application block can be used to implement interception policies to streamline the implementation of common features, such as logging, caching, exception handling, and validation, across a system.

Security Application Block: Developers can use this application block to incorporate authorization and security caching functionality in their applications.

Unity Application Block: Developers can use this application block as a lightweight and extensible dependency injection container with support for constructor, property, and method call injection, as well as instance and type interception.

Validation Application Block: Developers can use this application block to create validation rules for business objects that can be used across different layers of their applications.

For more information on Microsoft Enterprise Library, refer to:
<http://msdn.microsoft.com/en-us/library/ff648951.aspx>



CelloSaaS uses Microsoft Application blocks Ver 4.0 and recommends the developers also use the same, upgrading or downgrading these block may cause serious problems or some of the framework components might also stop working. In future, these blocks will be upgraded in the Framework, so developers are advised to hold until that, please check the product roadmap or contact the CelloSaaS support.

4.1 Foundation (Plumbing) Features

CelloSaaS framework consists of plumbing layer components, building blocks, and various SaaS specific components, allowing a 30% reduction in developing a SaaS based solution. Large portion of the developer's time are spent writing "plumbing" code for common services, rather than building



business logic specific to the application. CelloSaaS framework provides the common application functionality that you do not want to “reinvent”. It lets you focus on your business logic.

When it comes to SaaS, the plumbing layer is critical as it brings in the elements of security, performance, etc. together. This will set the base for future scalability of the product as the user base and product grows. So, developers tend to focus more on making this plumbing layer really strong before they can actually venture into building SaaS application. This guide helps you to quickly grasp what enterprise library can do for you, present examples, and makes it easier for you to start experimenting. The guide will walk you through the most common usage scenarios for each of the following application blocks, each aimed at managing specific cross-cutting concerns. Most common usage scenarios include the following:

Improving performance by utilizing a local in-memory or isolated storage cache.

Calling into your database stored procedures and managing the results exposed as a sequence of objects for client side querying.

Incorporating cryptography mechanisms to protect your data.

Designing and implementing a consistent strategy for managing exceptions that occur in various architectural layers of your application.

Implementing system logging through the wide variety of out-of-the box logging sinks or your custom provider.

Performing structured and easy-to-maintain validation using attributes and rules sets.

4.2 Model-View-Controller (MVC)

CelloSaaS is built on top of ASP.NET MVC and hence inherently follows the MVC pattern. The model is used to manage information, organize a project for easy development and maintenance while protecting your data and reputation. MVC pattern is applied to the server code of web applications. This pattern works well in web applications because the HTTP verb + URL combination is a very good way to determine which action to take. Your application will be finished quicker with higher quality. MVC based CelloSaaS allows you to separate model (basic classes that represent domain you are working with), controller, and views.

When you use a CelloSaaS framework, the basic structure for MVC is already prepared and you just have to extend that structure, placing your files in the appropriate directory, to comply with the Model-View-Controller pattern. Also you get a lot of functionality already written and thoroughly tested.

If you have installed Visual Studio 2010, ASP.NET MVC is already installed on your computer. If you have installed Visual Studio 2008, download ASP.NET MVC from Microsoft Download Center.

<http://www.asp.net/mvc/mvc3>

4.3 Exception handling

Exceptions are the standard mechanism for reporting errors and applications and libraries should not use return codes to communicate errors. The use of exceptions adds to a consistent framework design and allows error reporting from members, such as constructors, that cannot have a return type. Exceptions also allow programs to handle the error or terminate as appropriate. The default behavior is to terminate an application if it does not handle a thrown exception. CelloSaaS offers exception handling components to handle different kinds of exceptions. This is built on top of Enterprise Library.

4.3.1 Exception handling Class Members

When an exception is thrown, HandleException method from CelloSaaS framework handles the error.

Method

```
bool HandleException(Exception exception, string policyName)
```

Handles general exception and calls the exception handlers specified for the policymame. If the policy is not passed, then the exception is logged using default policy.

```
bool HandleException(BaseException exception, string policyName)
```

Handles base exception and calls the exception handlers specified for the policymame. If the policymame is null, sets the policymame by getting the exception handling setting and determines the policy based on the severity of the exception.

Parameters

Exception - Specifies general exception code and the exception details.

BaseException – Specifies user defined module exception.

PolicyName - Specifies the Exception Logger.Default PolicyName:" GlobalExceptionLogger".

Return value - Bool - Boolean- Specifies if the exception needs to be rethrown.

4.3.2 How is Exception handled in CelloSaaS?

The sample code below clearly explains how exception is handled in CelloSaaS.

```
Try
{
}
catch (ArgumentException argumentException)
{
    ExceptionService.HandleException(argumentException, _defaultPolicy);
    ModelState.AddModelError("login", Resources.AccountResource.e_ParameterEmptyOrNull);
}
catch (UserDetailException exception)
{
    ExceptionService.HandleException(exception, _defaultPolicy);
    ModelState.AddModelError("login", Resources.AccountResource.e_userDetails);
}
```



<>policyName>> Specifies name of the policy as described in the application configuration file under <exceptionPolicies>tag as shown below.

```
<ExceptionHandling>
<ExceptionPolicies>
    <add name="GlobalExceptionLogger">
        <ExceptionTypes>
            <add type="System.Exception, mscorelib, Version=2.0.0.0, Culture=neutral,
PublicToken=b77a5c561934e089" postHandlingAction="None" name="Exception">
                <exceptionHandlers>
                    <add logCategory="General" eventId="100" severity="Error" title="Enterprise Library
Exception Handling"
formatterType="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.XmlExceptionFormatter,
Microsoft.Practices.EnterpriseLibrary.ExceptionHandling, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" priority="0"
type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.LoggingExceptionHandler,
Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" name="Logging Handler"/>
                </exceptionHandlers>
            </add>
        </ExceptionTypes>
    </add>
```

```
</exceptionTypes>
</add>
<add name="EventSchedulerExceptionLogger">
<exceptionTypes>
<add type="System.Exception, mscorel, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" postHandlingAction="None" name="Exception">
<exceptionHandlers>
<add logCategory="EventExceptionLog" eventId="101" severity="Error" title="Event
Scheduler Exception Handling"
formatterType="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.XmlExceptionFormatter,
Microsoft.Practices.EnterpriseLibrary.ExceptionHandling, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" priority="0"
type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.LoggingExceptionHandle
r, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" name="Event Scheduler Logging
Handler"/>
</exceptionHandlers>
</add>
</exceptionTypes>
</add>
<add name="ezCLMExceptionPolicy">
<exceptionTypes>
<add type="System.Exception, mscorel, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" postHandlingAction="None" name="Exception">
<exceptionHandlers>
<add logCategory="ezCLMLog" eventId="102" severity="Error" title="ezCLM Exception
Handling"
formatterType="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.XmlExceptionFormatter,
Microsoft.Practices.EnterpriseLibrary.ExceptionHandling, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" priority="0"
type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.LoggingExceptionHandle
r, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" name="ezCLMExceptionPolicy" />
</exceptionHandlers>
</add>
</exceptionTypes>
</add>
</exceptionPolicies>
</exceptionHandling>
```

4.4 Service Post and Preprocessors

There are several instances where the developers need to do some pre-processing and post processing operations while doing database related CRUD operations. For example, consider a CRM system and assume that there needs to be a notification to the super user whenever there is a customer deletion. This notification needs to be sent as a part of post processing. And, there would be several such instances in developing a SaaS application. CelloSaaS offers Post and Preprocessors which are used for invoking loosely coupled integration points and process that needs to be called before or after your action item such as insert, update, delete. There are two types of processors:

Preprocessors

Postprocessors

Preprocessors

This is used to define actions that need to be performed before any CRUD operation. In order to define a Preprocessor, you have to first create a class file and inherit from interface called IPreProcessorProvider, which is available in CelloSaaS.Services.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using CelloSaaS.Model;
namespace CelloSaaS.Services
{
public interface IPreProcessorProvider
{
    bool PreProcessorInsert(object entity, params object[] args);
    bool PreProcessorUpdate(string referenceId, object entity, params object[] args);
    bool PreProcessorDelete(string referenceId, object entity, params object[] args);
    bool PreProcessorPermanentDelete(string referenceId, object entity, params object[] args);
}
}
```

Postprocessors

This is used to define actions that need to be called after CRUD operations. In order to define a Postprocessor you have to first create a class file and inherit from interface called IPostProcessorProvider, which is available in CelloSaaS.Services.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using CelloSaaS.Model;
namespace CelloSaaS.Services
{
public interface IPostProcessorProvider
{
    bool PostProcessorInsert(string referenceId, object entity, params object[] args);
    bool PostProcessorUpdate(string referenceId, object entity, params object[] args);
    bool PostProcessorDelete(string referenceId, object entity, params object[] args);
    bool PostProcessorPermanentDelete(string referenceId, object entity, params object[] args);
}
}

<TenantProvisionSetting defaultProvider="Default">
<PreProcess>
<add name="Default" assembly="WebApplication"
type="WebApplication.Models.TenantValidationPreProcessor" />
</PreProcess>
<PostProcess>
<add name="Default" assembly="CelloSaaS.Services"
type="CelloSaaS.Services.TenantManagement.DefaultScriptPostProcess" />
</PostProcess>
</TenantProvisionSetting>>
```

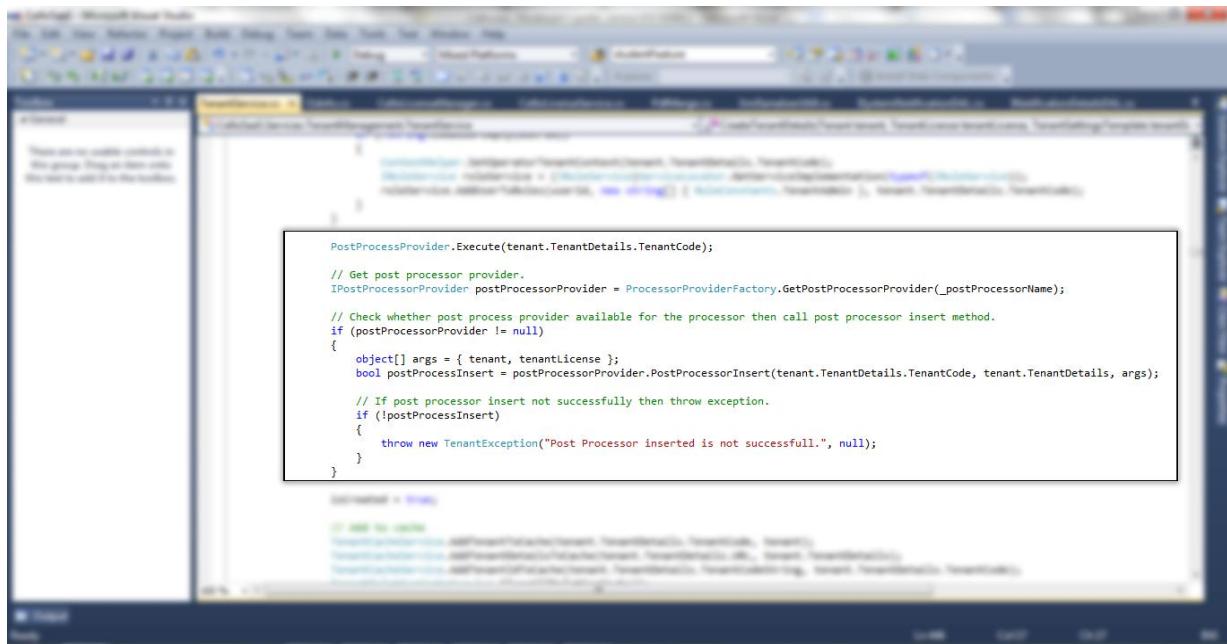


Figure 4-1 – PostProcessor Provider Screen

4.5 Scalable Connection Strings Management

CelloSaaS allows the application to scale horizontally and be partitioned based on tenant at a database level. To enable this, all the connection strings in the application should be accessed via CelloSaaS methods. There should be at least one connection string defined with the name applicationconnection string. This is the default connection string for the application.

To access the CelloSaaS connection string, use the following method:

CelloSaaS.DataAccessLayer.DBMetaData.GetCelloSaaSConnectionString()

To access other named connection string, for instance a module connection string use the following method:

CelloSaaS.DataAccessLayer.DBMetaData.GetConnectionString(string connectionStringName)

The type of connection string provider can be configured in the web.config using the section. The connection string provider needs to be configured in the web.config.

```

<dalConnectionStringManager defaultProvider="Default">
<providers>
<add name="Default" assembly="CelloSaaS.Library" type="CelloSaaS.Library.Provider.DALConnectionStringProvider"/>
</providers>
</dalConnectionStringManager>

```

DALConnectionStringProvider is a default provider which will be taking connection string from web config. CelloSaaS provides PartitionDALConnectionStringProvider which is used to allow application to scale out horizontally.

```
<dalConnectionStringManagerdefaultProvider="Default">
<providers>
<addname="Default"assembly="CelloSaaS.Library" type="CelloSaaS.Services.Providers.PartitionDALConnectionStringProvider"/>
</providers>
</dalConnectionStringManager>
```

PartitionDALConnectionStringProvider will take the connection strings from the tenant settings. Settings value must have connectionstring and Provider with the delimiter (~).

Example

```
Data Source=ServerName;Initial Catalog=DBName;User Id=UserName;Password=Password;~System.Data.SqlClient
```

4.6 Logging

Logging is an application block in Microsoft Enterprise Library, which is re-usable and extensible source code-based guidance that simplifies development of common logging functionality in SaaS framework applications. Developers can use the logging block to write information to a variety of locations (such as event log, e-mail message, database, message queue, text file, and many others). The application block provides a consistent interface for logging information to any destination. Application code does not specify the destination for the information. Configuration settings determine whether the application block writes the logging information and the location of that information. This means that operators as well as developers can modify logging behavior without changing application code.

4.6.1 Logging Class Members

When a log event is invoked in CelloSaaS code, it is handled by the Write method.

Namespace: CelloSaaS.Library;

Class: LogService

Method - Static voidWrite(string category, string message);

Parameters

Category - specifies the category for which the policy should be used.

Message - specifies the message, writes the message to the log by calling log handler as configured for the specified category.

Return Value - Null

Static voidWrite(string message);

Parameters

Message - specifies the message, writes the message to the log by calling log handler by default category.

Return Value - Null

4.6.2 How is logging handled in CelloSaaS?

```
try
{
//Calling service for writing message to default category
LogService.Write("<<message>>");
//Calling service for writing message to specified category
LogService.Write("<<category>>","<<message>>");
}
```



<<message>> Specifies the message to be entered in log file. <<category>> specifies the category for which the policy should be used.

Logging has to be configured in Web.Config file as follows.

```
<loggingConfigurationname="Logging Application
Block"tracingEnabled="true"defaultCategory="General"logWarningsWhenNoCategoriesMatch="true"
>
<listeners>
<addsource="Enterprise Library Logging"formatter="Text
Formatter"log="Application"machineName=""listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.FormattedEventLogTraceListenerData,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
traceOutputOptions="None"type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.FormattedEventLogTraceListener, Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"name="Formatted EventLog TraceListener"/>

<addname="Flat File
Destination"type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.FlatFileTraceListener, Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.FlatFileTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"fileName="UnHandledExceptions.log"/>
</listeners>
<categorySources>
    <addswitchValue="All"name="General">
        <listeners>
            <addname="Flat File Destination"/>
            <addname="Formatted EventLog TraceListener"/>
        </listeners>
        </add>
    </categorySources>
    </loggingConfiguration>
```

4.7 Scheduler

CelloSaaS offers two types of schedulers:

Event based Schedulers

Time based Schedulers

4.7.1 CelloSaaS Event System

CelloSaaS Event system provides the application to run various jobs as per the tenants need in the occurrence of the system event. “System Event” is any event of interest that happens in the application that requires special attention, like Addition / Update / removal of an Employee, order confirmation etc. The developer only needs to identify various events that must be raised in the system and uses the CelloSaaS Event API to raise them. At runtime the tenant admin can configure what job must run when the event happens, and this is configurable via the user interface. The various jobs that the tenant may require may be thought through and should register it with CelloSaaS so that the tenant can pick one.

By default CelloSaaS provides you with a Job named “Workflow Creation Job” which can be mapped to any event through the UI. Once mapped to an Event this job will create a workflow instance and start it (More information is on the section “Event Workflow handler”).

Note: Events are global to all tenants. Only the jobs mapped to an event are tenant based.

Event APIs:

```
interface CelloSaaS.EventScheduler.ServiceContracts.IEventMetadataService
{
    /// <summary>
    /// Gets all event meta data.
    /// </summary>
    /// <param name="tenantId">The tenant id.</param>
    /// <returns></returns>
    /// <exception cref="CelloSaaS.EventScheduler.ServiceContracts.EventActivityException"></exception>
    Dictionary<string, EventMetadata> GetAllEventMetadata(string tenantId);

    /// <summary>
    /// Gets the event meta data by event id.
    /// </summary>
    /// <param name="eventId">The event id.</param>
    /// <param name="tenantId">The tenant id.</param>
    /// <returns></returns>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception cref="CelloSaaS.EventScheduler.ServiceContracts.EventActivityException"></exception>
    EventMetadata GetEventMetadataById(string eventId, string tenantId);

    /// <summary>
    /// Adds the event meta data.
    /// </summary>
    /// <param name="eventMetadata">The event metadata.</param>
    /// <returns></returns>
    /// <exception cref="System.ArgumentException"></exception>
```



```
/// <exception
cref="CelloSaaS.EventScheduler.ServiceContracts.DuplicateEventNameException"></exception>
/// <exception
cref="CelloSaaS.EventScheduler.ServiceContracts.EventActivityException"></exception>
string AddEventMetadata(EventMetadata eventMetadata);

/// <summary>
/// Deletes the event meta data.
/// </summary>
/// <param name="eventId">The event id.</param>
/// <exception cref="System.ArgumentNullException"></exception>
/// <exception
cref="CelloSaaS.EventScheduler.ServiceContracts.EventActivityException"></exception>
void DeleteEventMetadata(string eventId);

/// <summary>
/// Updates the event meta data.
/// </summary>
/// <param name="eventMetadata">The event metadata.</param>
/// <exception cref="System.ArgumentException"></exception>
/// <exception
cref="CelloSaaS.EventScheduler.ServiceContracts.DuplicateEventNameException"></exception>
/// <exception
cref="CelloSaaS.EventScheduler.ServiceContracts.EventActivityException"></exception>
void UpdateEventMetadata(EventMetadata eventMetadata);

}
```

Developers can use the above API to create an Event in CelloSaaS. It can also be done through User Interface via Product Admin login. This is one time process.

Tip: Create necessary events via API or UI and create the below class to hold the event identifier which will be later used to raise the event.

```
public class EventConstants
{
    public const string EmployeeAdded = "FD9E31D8-EA53-40DE-9453-147DF9BB5390";
}
```

To raise the events in the system

```
interface IEventRegister
{
    /// <summary>
    /// This method should be called when any events should be raised in any part of the system.
    /// </summary>
    /// <param name="applicationEvent">Application event parameter</param>
    void RegisterEvent(Event applicationEvent);
}
```

Sample

The following sample demonstrated on how to raise an event when any employee is created.

```
public void AddEmployee(Employee entity)
{
    // your code to create an employee
    // once successful then raise the event
```



```
EventRegisterProxy.RegisterEvent(new Event
{
    EventId = EventConstants.EmployeeAdded,
    TenantId = UserIdentity.TenantID,
    UserId = UserIdentity.UserId,
    SubjectType = "Employee",
    SubjectXmlValue = entity.SerializeToXml()
});
}
```

When the above code is executed the CelloSaaS Processing Engine will run the jobs configured by the Tenant.

The jobs are queued in database and run in batch mode according to the priority by the CelloSaaS Event Scheduler Windows Service to improve the performance. Batch mode means, it will process new events of size (5000) set in the configuration file at any given time. All the events / jobs are audited by CelloSaaS and are available through User Interface

Navigation

Admin -> Audits->Event Audits and Admin -> Audits -> Job Audits and through the APIs.

The Event Model contains the following properties which facilitate persisting application objects to be used in content templates to replace the placeholders.

SubjectId
SubjectValue/SubjectXmlValue
ContextId
ContextValue/ContextXmlValue
TargetId
TargetValue/TargetXmlValue

InProc Environment

SubjectValue / ContextValue / TargetValue can contain any object.

Web Service Environment

SubjectXmlValue / ContextXmlValue / TargetXmlValue will contain the serialized values of your business object. So persisting them is easy and the end user needs to use XPath expression to replace the placeholders in the template content, can be used in both WCF and InProc environment.

SubjectId / ContextId / TargetId contain user friendly string to facilitate easy searching by the end user in UI.

The events can also be mapped to a content template by the tenant via user interface at runtime. This enables the event audit log to be used like Activity Stream. So instead of viewing raw event audit, you can view meaningful information.

```
/// <summary>
/// Represents the Event details that is
/// passed when registering event with EventRegister
/// </summary>
public class Event
{
    /// <summary>
```



```
/// Gets or sets the identifier.  
/// </summary>  
/// <value>  
/// The identifier.  
/// </value>  
public string Identifier { get; set; }  
  
/// <summary>  
/// Gets or sets the event id.  
/// </summary>  
/// <value>The event id.</value>  
public string EventId { get; set; }  
  
/// <summary>  
/// Gets the event description.  
/// </summary>  
/// <value>The event description.</value>  
public string EventDescription { get; set; }  
  
/// <summary>  
/// Gets or sets the user id.  
/// </summary>  
/// <value>The user id.</value>  
public string UserId { get; set; }  
  
/// <summary>  
/// Gets the transformed message.  
/// </summary>  
/// <value>The message.</value>  
public string TransformedDescription { get; set; }  
  
/// <summary>  
/// Gets or sets the notification event parameter.  
/// </summary>  
/// <value>The notification event parameter.</value>  
public IEventParameter NotificationEventParameter { get; set; }  
  
/// <summary>  
/// Gets or sets the notification EventParameter Xml  
/// </summary>  
/// <value>  
/// notification eventParameter Xml  
/// </value>  
public string EventParameterXml { get; set; }  
  
/// <summary>  
/// Gets or sets the tenant id.  
/// </summary>  
/// <value>  
/// The tenant id.  
/// </value>  
public string TenantId { get; set; }  
  
/// <summary>  
/// Gets or sets the tenant code. [friendly name]  
/// </summary>  
/// <value>  
/// The tenant code.
```



```
/// </value>
public string TenantCode { get; set; }

/// <summary>
/// Gets or sets the priority.
/// </summary>
/// <value>
/// The priority.
/// </value>
public EventPriority Priority { get; set; }

/// <summary>
/// Gets or sets the SubjectId.
/// </summary>
/// <value>The SubjectId.</value>
public string SubjectId { get; set; }

/// <summary>
/// Gets or sets the SubjectType.
/// </summary>
/// <value>The SubjectType.</value>
public string SubjectType { get; set; }

/// <summary>
/// Gets or sets the SubjectValue. Do not set the values in wcf services.
/// </summary>
/// <value>The SubjectValue.</value>
public object SubjectValue { get; set; }

/// <summary>
/// Gets or sets the subject XML value.
/// </summary>
/// <value>The subject XML value.</value>
public string SubjectXmlValue { get; set; }

/// <summary>
/// Gets or sets the TargetId.
/// </summary>
/// <value>The TargetId.</value>
public string TargetId { get; set; }

/// <summary>
/// Gets or sets the TargetType.
/// </summary>
/// <value>The TargetType.</value>
public string TargetType { get; set; }

/// <summary>
/// Gets or sets the TargetValue. Do not set the values in wcf services.
/// </summary>
/// <value>The TargetValue.</value>
public object TargetValue { get; set; }

/// <summary>
/// Gets or sets the target XML value.
/// </summary>
/// <value>The target XML value.</value>
public string TargetXmlValue { get; set; }
```



```
/// <summary>
/// Gets or sets the ContextId.
/// </summary>
/// <value>The ContextId.</value>
public string ContextId { get; set; }

/// <summary>
/// Gets or sets the ContextType.
/// </summary>
/// <value>The ContextType.</value>
public string ContextType { get; set; }

/// <summary>
/// Gets or sets the ContextValue. Do not set the values in wcf services.
/// </summary>
/// <value>The ContextValue.</value>
public object ContextValue { get; set; }

/// <summary>
/// Gets or sets the context XML value.
/// </summary>
/// <value>The context XML value.</value>
public string ContextXmlValue { get; set; }

/// <summary>
/// Gets or sets the CreatedOn.
/// </summary>
/// <value>The CreatedOn.</value>
public DateTime CreatedOn { get; set; }

/// <summary>
/// Gets or sets the CreatedBy.
/// </summary>
/// <value>The CreatedBy.</value>
public string CreatedBy { get; set; }

/// <summary>
/// Record status
/// </summary>
/// <value>true or false</value>
public bool Status { get; set; }

/// <summary>
/// Gets or sets the event status.
/// </summary>
/// <value>
/// The event status.
/// </value>
public EventStatus EventStatus { get; set; }

/// <summary>
/// Gets or sets the EventName.
/// </summary>
/// <value>The EventName.</value>
public string EventName { get; set; }

/// <summary>
```



```
/// Gets or sets the TemplateId.  
/// </summary>  
/// <value>The TemplateId.</value>  
public string TemplateId { get; set; }  
  
}  
/// <summary>  
/// Describes Event execution priority  
/// </summary>  
  
public enum EventPriority  
{  
    /// <summary>  
    /// Execution priority is low  
    /// </summary>  
    Low,  
    /// <summary>  
    /// Execution priority is above low and below high  
    /// </summary>  
    Medium,  
    /// <summary>  
    /// Highest execution priority  
    /// </summary>  
    High  
}  
  
/// <summary>  
/// Describes Event execution Status  
/// </summary>  
public enum EventStatus  
{  
    /// <summary>  
    /// New denote newly entered entry not yet processed.  
    /// </summary>  
    New,  
  
    /// <summary>  
    /// Queued denote this is added to some queue and waiting to be processed.  
    /// </summary>  
    Queued,  
  
    /// <summary>  
    /// InProgress denotes this is send to the dispatcher (In-Execution).  
    /// </summary>  
    InProgress,  
  
    /// <summary>  
    /// Success denotes this is processed successfully.  
    /// </summary>  
    Success,  
  
    /// <summary>  
    /// Failure denotes failure to execute.  
    /// </summary>  
    Failure  
}
```

Event Template Mapping APIs

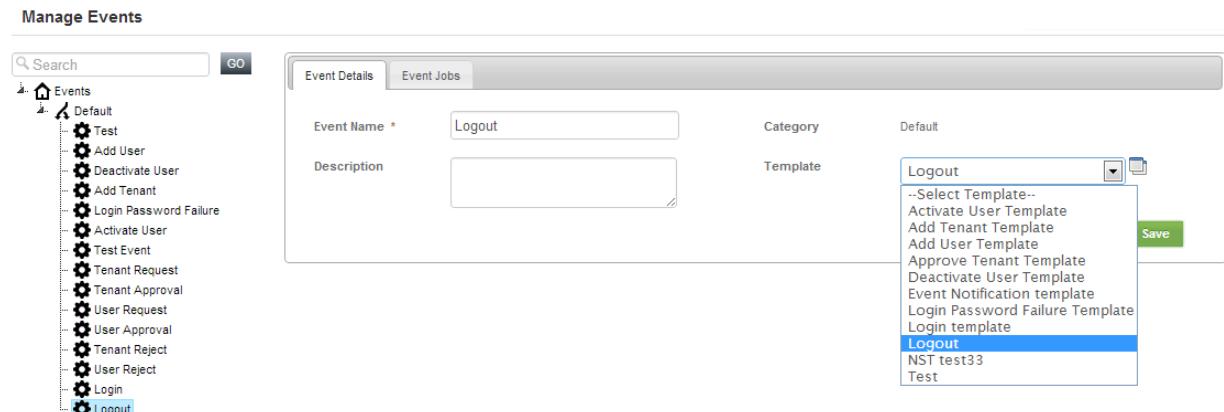
```

interface CelloSaaS.EventScheduler.ServiceContracts.IEventTemplateService
{
    /// <summary>
    /// Mapping Event to Template
    /// </summary>
    /// <param name="eventId">The event id.</param>
    /// <param name="templateId">The template id.</param>
    /// <param name="tenantId">The tenant id.</param>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception cref="CelloSaaS.EventScheduler.Model.EventTemplateException"></exception>
    void MapEventTemplate(string eventId, string templateId, string tenantId);

    /// <summary>
    /// Get the template id based on event id and tenant id
    /// </summary>
    /// <param name="eventId">The event id.</param>
    /// <param name="tenantId">The tenant id.</param>
    /// <returns></returns>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception cref="CelloSaaS.EventScheduler.Model.EventTemplateException"></exception>
    string GetEventTemplate(string eventId, string tenantId);
}

```

Mapping Content Template to an Event



The screenshot shows the 'Manage Events' interface. On the left, there's a tree view under 'Events' with several items like 'Default', 'Test', 'Add User', etc. In the center, there's a form for 'Event Details'. The 'Event Name' field contains 'Logout'. The 'Category' field is 'Default'. The 'Template' field has a dropdown menu open, showing a list of templates. The 'Logout' template is selected and highlighted in blue. A 'Save' button is visible at the bottom right of the dropdown.

[Mapping Content Template to an Event]

Creating and Registering Job

To Create custom Jobs follow the below steps.

Create a class inheriting from CelloSaaS.EventScheduler.EventPublishingEngine.IJob.

```

public class SampleJob : CelloSaaS.EventScheduler.EventPublishingEngine.IJob
{
    public void Execute(Event eventParameter)
    {
        // write your business logic here
    }
}

```



```
}
```

Add an entry to [Jobs] table.

```
INSERT INTO [dbo].[Jobs]
([Job_Name],[Job_Type],[Job_Description],[Job_CreatedBy],[Job_CreatedOn],[Job_Status])
VALUES
('Sample Job', 'SampleApp.SampleJob,SampleApp','Job description','3398f837-b988-4708-999d-d3dfe11875b3',GETDATE(),1)
```

Job_Type Value should be in this format: [FullNamespace.TypeName, AssemblyName]

Note

CelloSaaS provides default job to create work flow instance. Below script is for the default WFEEventHandler Job.

```
INSERT INTO [dbo].[Jobs]
([Job_Id],[Job_Name],[Job_Type],[Job_Description],[Job_CreatedBy],[Job_CreatedOn],[Job_Status])
)
VALUES
('ab860f9a-74b4-e111-98c8-000000000000','Workflow Creation
Job','CelloSaaS.DefaultActivities.WFEEventHandler,CelloSaaS.DefaultActivities','This job will create a workflow instance.','3398f837-b988-4708-999d-d3dfe11875b3',GETDATE(),1)
```

APIs

To create, delete, fetch the Job details

```
interface CelloSaaS.EventScheduler.ServiceContracts.ISchedulerService
{

    /// <summary>
    /// This service will add a new job record
    /// </summary>
    /// <param name="job">job details to be added</param>
    /// <returns>returns the newly created job identifier</returns>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception cref="CelloSaaS.EventScheduler.ServiceContracts.JobException"></exception>
    string AddJob(Job job);

    /// <summary>
    /// This service will delete the given job id
    /// </summary>
    /// <param name="jobId">job identifier to delete</param>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception cref="CelloSaaS.EventScheduler.ServiceContracts.JobException"></exception>
    void DeleteJob(string jobId);

    /// <summary>
    /// This method fetches the job details for the given job id
    /// </summary>
    /// <param name="jobId">job identifier</param>
    /// <returns>returns the job details if found or null is returned</returns>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception cref="CelloSaaS.EventScheduler.ServiceContracts.JobException"></exception>
    Job GetJob(string jobId);
```

}

Mapping Jobs to Event

Use the below APIs to map jobs to events. CelloSaaS provides a default Workflow Job which can be mapped to any Event via UI. If necessary the application developer can modify the user interface to display/restrict the available jobs to the user.

```
interface CelloSaaS.EventScheduler.ServiceContracts.IEventSchedulerService
{
    /// <summary>
    /// Adds the given job to the event.
    /// </summary>
    /// <param name="eventId">The event id.</param>
    /// <param name="jobId">The job id.</param>
    /// <param name="tenantId">The tenant id.</param>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception
        cref="CelloSaaS.EventScheduler.ServiceContracts.EventScheduleException"></exception>
    void AddEventJob(string eventId, string jobId, string tenantId);

    /// <summary>
    /// Deletes the job from the event.
    /// </summary>
    /// <param name="eventId">The event id.</param>
    /// <param name="jobId">The job id.</param>
    /// <param name="tenantId">The tenant id.</param>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception
        cref="CelloSaaS.EventScheduler.ServiceContracts.EventDetailsException"></exception>
    void DeleteEventJob(string eventId, string jobId, string tenantId);

    /// <summary>
    /// This method will search for the jobs mapped to the given event.
    /// </summary>
    /// <param name="eventIds">array of event identifier</param>
    /// <returns>returns found jobs in dictionary object</returns>
    /// <exception cref="System.ArgumentNullException"></exception>
    /// <exception cref="CelloSaaS.EventScheduler.ServiceContracts.JobException"></exception>
    Dictionary<string, Job> GetJobs(string eventId, string tenantId);
}
```

Event and Job Audit APIs

```
interface CelloSaaS.EventScheduler.ServiceContracts.IEventAuditService
{
    /// <summary>
    /// Get ActivityEventLog details by eventId,tenantid
    /// </summary>
    /// <param name="eventId">The event id.</param>
    /// <param name="tenantId">The tenant id.</param>
    /// <returns>ActivityEventLog</returns>
    /// <exception cref="System.ArgumentNullException"></exception>
```



```
/// <exception cref="CelloSaaS.EventScheduler.Model.ActivityEventLogException"></exception>
Dictionary<string, Event> GetEventAudits(string eventId, string tenantId);

/// <summary>
/// Get all ActivityEventLog details for the tenantId.
/// </summary>
/// <param name="userId">The user id.</param>
/// <returns>Dictionary</returns>
/// <exception cref="System.ArgumentNullException"></exception>
/// <exception cref="CelloSaaS.EventScheduler.Model.ActivityEventLogException"></exception>
Dictionary<string, Event> GetEventAuditsByUserId(string userId);

/// <summary>
/// Searches the event audit details.
/// </summary>
/// <param name="eventAuditSearchCondition">The event audit search condition.</param>
/// <returns></returns>
/// <exception cref="System.ArgumentNullException"></exception>
/// <exception cref="System.ArgumentException"></exception>
/// <exception cref="CelloSaaS.EventScheduler.Model.ActivityEventLogException"></exception>
EventAuditSearchResult GetEventAuditDetails(EventAuditSearchCondition
eventAuditSearchCondition);

}

interface CelloSaaS.EventScheduler.ServiceContracts.IJobAuditService
{

/// <summary>
/// Searchs the job audits with the given conditions.
/// </summary>
/// <param name="searchCondition">search condition</param>
/// <returns></returns>
/// <exception cref="System.ArgumentException"></exception>
/// <exception cref="System.ArgumentNullException"></exception>
/// <exception cref="CelloSaaS.EventScheduler.ServiceContracts.JobAuditException"></exception>
JobAuditSearchResult SearchJobAudits(JobAuditSearchCondition searchCondition);

}
```

The returned **Event** object will contain the template content if mapped in the “TransformedDescription” property. The template content text can contain place holders which will be replaced with the object value. More information on placeholders is in the Content Template Section.

How to Guide: [Implementing Event Audit](#)

Event Workflow Job Handler

CelloSaaS by default provides a Job named “**Workflow Creation Job**” which can be mapped to any event via UI or API. Once mapped then whenever the event occur it creates the given Workflow Instance and starts the workflow.

To map Workflow Creation Job to an event

```
EventSchedulerProxy.AddEventJob(eventId, "ab860f9a-74b4-e111-98c8-000000000000",
UserIdentity.TenantID);
WFJobParameter wfJobParam = new WFJobParameter();
wfJobParam.WFName = "Name of the workflow";
wfJobParam.MapIdXPath = "Xpath/to/Mapid";
```

Note:

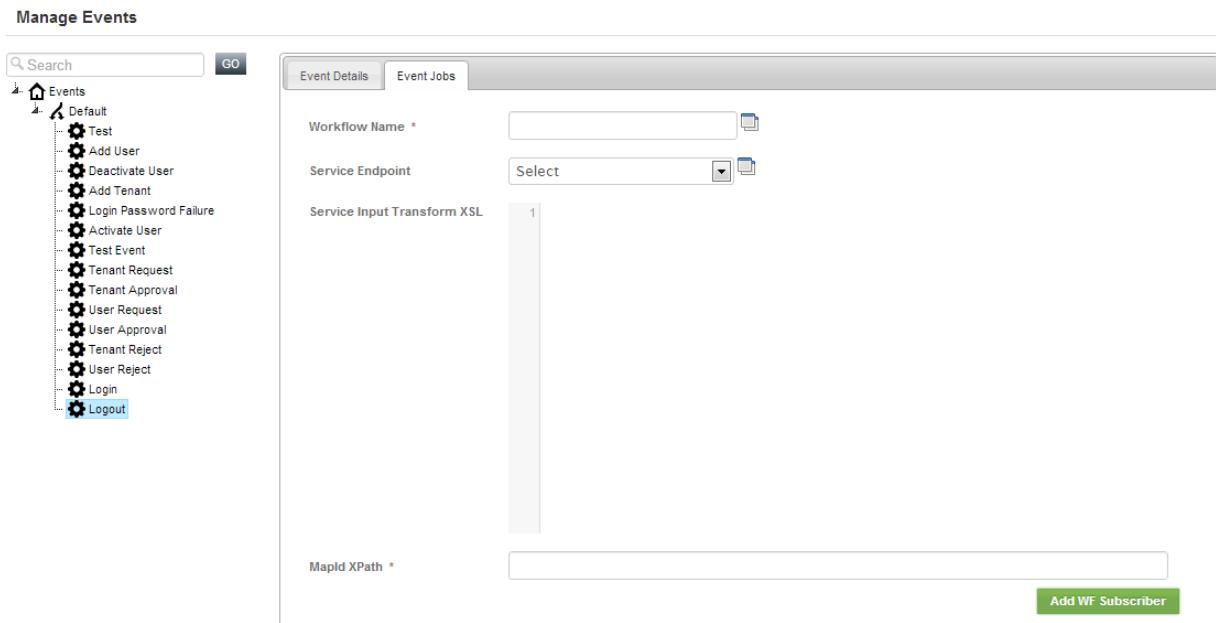
MapIdXPath will be obtained from the Event Xml. So mapId need to be passed when raising the event. eg. use SubjectId or ContextId or TargetId, etc to store the mapId.

```
var jparam = new JobParameters
{
    EventId = eventId,
    JobId = "ab860f9a-74b4-e111-98c8-000000000000",
    EventJobParameter = wfJobParam,
    TenantId = UserIdentity.TenantID,
    CreatedBy = UserIdentity.UserId,
    Status = true
};
```

```
JobParameterServiceProxy.AddJobParameter(jparam);
```

Instead of using the above code, default UI is provided to map this job.

Mapping Workflow Job to an Event



[Mapping Workflow Job to an Event.]

Passing Workflow input

To create a workflow instance you need a mapId and Wfinput object. The default WFEEventHandler Job will create the workflow instance using mapId got from MapIdXPath
wfinput object will be got from WFInputFinder associated with Workflow.



Create Workflow input finder class

```
public class TestWFInputFinder : IWorkFlowInputFinder
{
    public Dictionary<string, IWorkflowInput> GetMappingObjects(List<string> mapIds, string
workFlowId, string tenantId)
    {
        var result = new Dictionary<string, IWorkflowInput>();

        // your business logic to form IWorkflowInput

        return result;
    }
}
```

Associate this with your Workflow in database. [dbo].[WorkFlow].WFInputFinderType = "AssemblyName, FullTypeName";

Note: - More information on WFInputFinder refer http://techcello.com/downloads/how-to/how_to_wf_inputs_and_finders.pdf

You might need to add this assembly dll and all referenced dll's and configuration changes to CelloSaaS Event Windows Service.

So the WFEEventHandler gets the mapId from MapIdXPath and wfInput by calling GetMappingObjects() method and creates the workflow instance.

4.7.2 Time based Scheduler

Sometime or other, we might get situations functionalities which requires execution of certain jobs based on time. Example, sending a report every day at 8.30 p.m. Once the trigger associated with the job, then the background Quartz service running under different app domain will execute it accordingly.

Sample Code

```
TimeSchedulerService timeSchedulerService = newTimeSchedulerService();
SchedulerService schedulerService = newSchedulerService();
// Create a new Job
Job timeJob = newJob
{
    Name = "Time Job",
    CreatedBy = UserIdentity.UserId,
    JobType = "TestJobs.QuartzTestJob, TestJobs.dll",
    Status = true
};
// Add to database
string jobId = schedulerService.AddJob(timeJob);
// create a new time schedule
TimeSchedule timeSchedule = newTimeSchedule
{
    JobId = job.JobId,
    JobReferenceName = "Test Job RefName",
    ScheduleId = "85E3577B-831D-4A8D-B034-60EBB161EF6B",
    StartDateTime = DateTime.UtcNow.AddSeconds(10), // start after 10 seconds
    EndDateTime = null,
```

```
CreatedBy = UserIdentity.UserId,
Status = true
};

// Create job parameters
JobParameter jobParameter = new JobParameter();
jobParameter.Key = "FileName";
jobParameter.Value = "SHA1";
// Add Job Parameter to list
IList<JobParameter> jobParameters = newList<JobParameter>();
jobParameters.Add(jobParameter);
// Now add the time schedule and job parameters to the database
timeSchedulerService.AddTimeScheduleToQuartz(timeSchedule,
jobParameters, Quartz.SimpleTrigger.RepeatIndefinitely, TimeSpan.FromDays(1));
```

This will execute the referenced job everyday at the specified StartDate. The jobs will be run by the Quartz windows service which is installed separately.

Configuration for Quartz Service

Add the below line in your quartz.config of the Quartz windows service.

```
quartz.plugin.injectJobListner.type = CelloSaaS.TimeScheduler.QuartzSchedulerPlugin,
```

CelloSaaS.TimeScheduler

Sample Quartz.config file

```
quartz.scheduler.instanceName = ServerScheduler
quartz.threadPool.type = Quartz.Simpl.SimpleThreadPool, Quartz
quartz.threadPool.threadCount = 10
quartz.threadPool.threadPriority = Normal
quartz.jobStore.type = Quartz.Impl.AdоЮbStore.JobStoreTX, Quartz
quartz.jobStore.driverDelegateType = Quartz.Impl.AdоЮbStore.StdAdoDelegate, Quartz
quartz.jobStore.tablePrefix = QRTZ_
quartz.jobStore.dataSource = celloScheduler
quartz.dataSource.celloScheduler.connectionString = Data Source=sqlserver;Initial Catalog=QuartzDb;User ID=username;Password=password;
quartz.dataSource.celloScheduler.provider = SqlServer-20
quartz.plugin.injectJobListner.type = CelloSaaS.TimeScheduler.QuartzSchedulerPlugin,
CelloSaaS.TimeScheduler
```

Job dll and the following dll's have to be copied to the Quartz windows service installed folder.

```
CelloSaaS.EventScheduler.dll
CelloSaaS.Library.dll
CelloSaaS.TimeScheduler.dll
Common.Logging.dll
Common.Logging.Log4Net.dll
log4net.dll
Microsoft.Practices.EnterpriseLibrary.Common.dll
Microsoft.Practices.EnterpriseLibrary.Data.dll
Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.dll
Microsoft.Practices.EnterpriseLibrary.Logging.dll
Microsoft.Practices.EnterpriseLibrary.PolicyInjection.dll
Microsoft.Practices.ObjectBuilder2.dll
Microsoft.Practices.Unity.Configuration.dll
Microsoft.Practices.Unity.dll
```

Quartz.dll
Quartz.Server.Core.dll

Add the following configuration section in the Quartz Windows service Application config file.

Sample App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<configSections>
<sectionname="quartz" type="System.Configuration.NameValueSectionHandler, System,
Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
<sectionname="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
<sectionGroupname="common">
<sectionname="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging"
/>
</sectionGroup>
<sectionname="dalConnectionStringManager" type="CelloSaaS.Library.Configuration.DALConnectio
nStringProviderConfiguration, CelloSaaS.Library" />
<sectionname="loggingConfiguration" type="Microsoft.Practices.EnterpriseLibrary.Logging.Configura
tion.LoggingSettings, Microsoft.Practices.EnterpriseLibrary.Logging" />
<sectionname="exceptionHandling" type="CelloSaaS.Library.Configuration.ExceptionHandlingSetting
s, CelloSaaS.Library" />
</configSections>
<common>
<logging>
<factoryAdaptertype="Common.Logging.Log4Net.Log4NetLoggerFactoryAdapter,
Common.Logging.Log4net">
<argkey="configType" value="INLINE" />
<argkey="configFile" value="quartz_sched.log" />
<argkey="level" value="INFO" />
</factoryAdapter>
</logging>
</common>
<connectionStrings>
<addname="CelloSaaSConnectionString" connectionString="Data Source=serverip;Initial
Catalog=DatabaseName;User
ID=userid;Password=password;" providerName="System.Data.SqlClient" />
</connectionStrings>
<log4net>
<appendername="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
<layouttype="log4net.Layout.PatternLayout">
<conversionPatternvalue="%d [%t] %-5p %l - %m%n" />
</layout>
</appender>
<appendername="EventLogAppender" type="log4net.Appender.EventLogAppender">
<layouttype="log4net.Layout.PatternLayout">
<conversionPatternvalue="%d [%t] %-5p %l - %m%n" />
</layout>
</appender>
</root>
<levelvalue="INFO" />
<appender-refref="EventLogAppender" />
</root>
</log4net>
<dalConnectionStringManagerdefaultProvider="Default" />
```

```

<providers>
<addname="Default"assembly="CelloSaaS.Library"type="CelloSaaS.Library.Provider.DALConnectio
nStringProvider" />
</providers>
</dalConnectionStringManager>
<loggingConfigurationname="Logging Application
Block"tracingEnabled="true"defaultCategory="General"logWarningsWhenNoCategoriesMatch="true"
>
<listeners>
<addname="Event Scheduler Exception
Listener"type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.FlatFileTraceListener,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"listenerDataType="Microsoft.Practices.EnterpriseLibrary.Loggi
ng.Configuration.FlatFileTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"fileName="EventSchedulerExceptions.log"/>
</listeners>
<categorySources>
<addswitchValue="All"name="EventExceptionLog">
<listeners>
<addname="Event Scheduler Exception Listener"/>
</listeners>
</add>
</categorySources>
</loggingConfiguration>
<exceptionHandling>
<exceptionPolicies>
<addname="EventSchedulerExceptionLogger">
<exceptionTypes>
<addtype="System.Exception, mscorelib, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"postHandlingAction="None"name="Exception">
<exceptionHandlers>
<addlogCategory="EventExceptionLog"eventId="101"severity="Error"title="Event Scheduler
Exception
Handling"formatterType="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.XmlExceptionFor
matter, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"priority="0"type="Microsoft.Practices.EnterpriseLibrary.Excepti
onHandling.Logging.LoggingExceptionHandler,
Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"name="Event Scheduler Logging Handler"/>
</exceptionHandlers>
</add>
</exceptionTypes>
</add>
</exceptionPolicies>
</exceptionHandling>
</configuration>

```



The job type in this instance should implement Quartz.IJob interface.

Triggers can be created with nearly any combination of the following directives:
at a certain time of day (to the millisecond)

on certain days of the week

on certain days of the month

on certain days of the year

not on certain days listed within a registered Calendar (such as business holidays)

repeated a specific number of times

repeated until a specific time/date

repeated indefinitely

repeated with a delay interval

Jobs are given names by their creator and can also be organized into named groups. Triggers may also be given names and placed into groups, in order to easily organize them within the scheduler. Jobs can be added to the scheduler once, but registered with multiple Triggers.

Here's a quick snippet of code, that instantiates and starts a scheduler, and schedules a job for execution:

```
Using Quartz.NET
// construct a scheduler factory
ISchedulerFactory schedFact = new StdSchedulerFactory();
// get a scheduler
IScheduler sched = schedFact.GetScheduler();
sched.Start();
// construct job info
JobDetail jobDetail = new JobDetail("myJob", null, typeof(HelloJob));
// fire every hour
Trigger trigger = TriggerUtils.MakeHourlyTrigger();
// start on the next even hour
trigger.StartTimeUtc = TriggerUtils.GetEvenHourDate(DateTime.UtcNow);
trigger.Name = "myTrigger";
sched.ScheduleJob(jobDetail, trigger);
```

For further details, please refer <http://quartznet.sourceforge.net/features.html>

4.7.3 Notification Manager [Obsolete]

What is Notification system?

Notification management is used for sending notifications like E-Mail, Files to recipients. The content of mail, sender and recipient address, name of the host can be configured in config file. We can have multiple handlers such as EmailNotificationHandler for a notification based on the user requirement.

How to send Notification?

Notification needs to be configured in web.config. The below sample configuration explains how to configure a notification.

Notification Configuration

```
<notificationSection>
<notifications>
<addname="SendConfirmation" message="Hi, {{username}}&lt;br&gt; This is confirmation message.">
<notificationHandlers>
<addname="email" type="CelloSaaS.Library.Notifications.EmailNotificationHandler" subject="Send Confirmation" from="admin@company.com" to="admin@company.com" cc="admin@company.com" host="mail.techcello.com">
</add>
</notificationHandlers>
</add>
</notifications>
</notificationSection>
```

Adding a <notifications>tag requires the following attributes.

Name: Identifier of the notification

Message: Content of the mail in template format with some placeholders

Inside add tag, we have another node called <notificationHandlers>. We can add notification with the following attributes:

- name:** Name of the Handler
- type:** Type of the notification Handler
- subject:** Subject of mail
- from:** Sender address
- to:** Recipient address
- host:** Host of mail sender
- cc:** Secondary Recipient
- username:** Smtip User Name
- password:** Smtip Password

After configuring, we can use the following methods to send notifications through code.

Name Space : "CelloSaaS.Library.Notifications".

```
///<summary>
/// This method is useful when bulk notification needs to be sent
///</summary>
///<param name="keyValues">Any static replacement values for message</param>
///<param name="args">objects which consist of the values to replace the place holders</param>
publicvoid Notify(Dictionary<string, string> keyValues, paramsobject[] args)
///<summary>
/// This reads the notification messages from the configured storage
/// prepares the message by substituting the values of placeholders with the values in object
/// loops the configured notification hanlders and sends out the notification
///</summary>
///<param name="notificationName">Name identifier of the notification</param>
///<param name="keyValues">Any static replacement values for message</param>
///<param name="args">objects which consist of the values to replace the place holders</param>
publicvoid Notify(string notificationName, Dictionary<string, string> keyValues, paramsobject[] args)
///<summary>
/// This method is an overloaded version of the previous method.
/// This is called when the recipient list is not configured but is passed by the consumer
///</summary>
///<param name="notificationName">Name identifier of the notification</param>
///<param name="recipients">Recipient address</param>
```

```
///<param name="secondaryRecipients">Secondary Recipient</param>
///<param name="from">Sender address</param>
///<param name="keyValues">Any static replacement values for message</param>
///<param name="args">objects which consist of the values to replace the place holders</param>
public void Notify(string notificationName, string recipients, string secondaryRecipients, string from,
Dictionary<string, string> keyValues, params object[] args)
```

4.7.4 Notification Manager [Current]

The following are the capabilities of the Notification Manager,

Send single or bulk mails

Upload single or bulk files to the FTP Server

Merged and upload multiple PDF files into a single file to an FTP server

Send single or bulk System Notifications

Notification Manager has two components

- **Notification Manager API**
- **Notification Windows Service**

Notification Manager API

Configure notification details [Dispatch and Content details]

Send an E-mail / System Notification / file transfer via FTP.

Notification Manager User Interface

The following are the Steps to be followed in the user interface to create a new notification from the User Interface.

Notification Detail Configuration

The Notification details configuration page will be available for the user, who has the "View_Notification" Privilege.

☞ Click on Admin>Notification Management

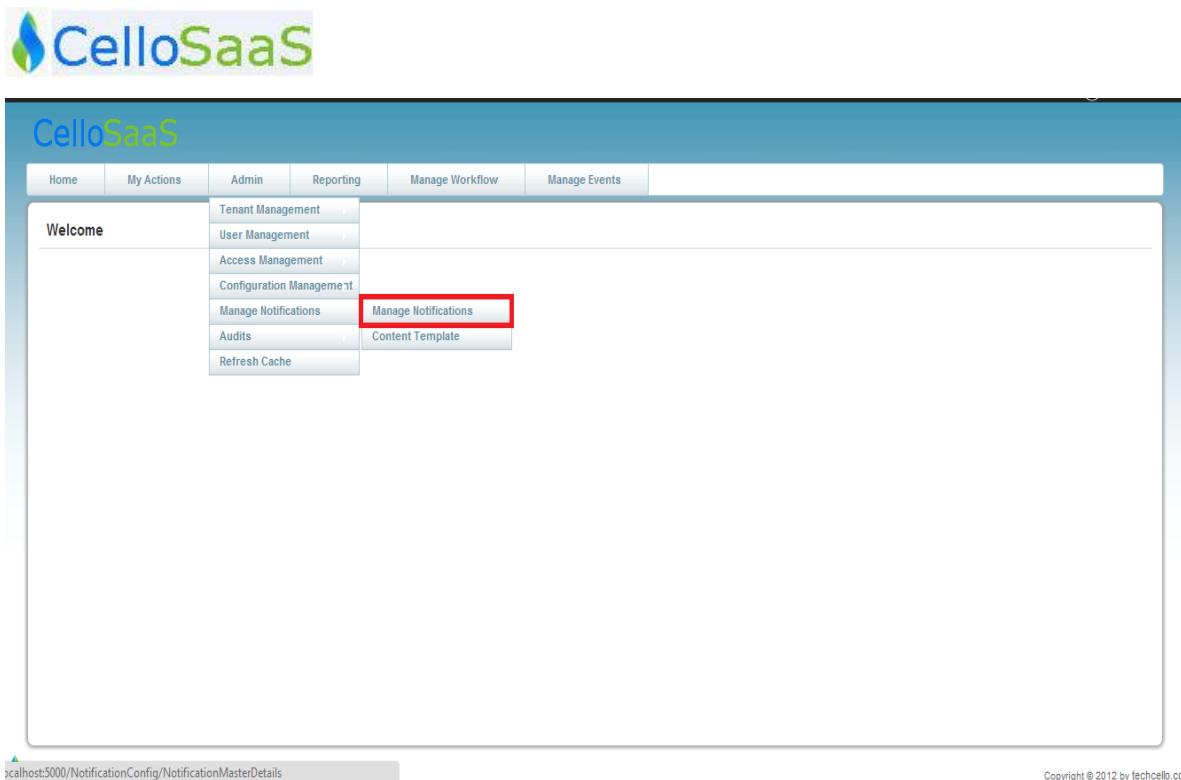


Figure 4-1 – Notification Management Menu

The screenshot shows the 'Manage Notification Details' screen. At the top, there is a search bar and a 'GO' button. Below the search bar is a table with the following data:

Notification Name	Description	Manage Dispatch	Edit Details	Delete Details
Forgot Password Mailer	Forgot Password Mailer			
User Confirmation Mailer	User Confirmation Mailer			
User Password Mailer	User Password Mailer			

At the bottom of the table, there is a pagination control showing 'Show 10 entries' and 'Showing 1 to 3 of 3 entries'.

Figure 4-2 – Manage Notification Details Screen

Notification Master Configuration

Notification master details are added as follows

Figure 4-3 – Manage Notification Details Screen

If the IsGlobal property is set to true, then the notification will be Global. Global notifications are available to all tenants. (i.e.) The content and dispatch details can be either overridden by the each tenant administrator or use the global details.

Once the notification master is added, click on manage dispatch icon to configure the dispatch and content details.

The notification dispatch and content details are configured as follows

Figure 4-4 – Manage Notification Details Screen

Email Configuration

Based on the Notification type selection, such as Email or Batch Email, the appropriate email related properties such as dispatch, content details will be shown as follows.



Approve Leave Notification

Notification Type: Email

Email Dispatch Details

Sender Address*	sender@host.com	Smtp Address*	mail.host.com
Smtp Username*	username	Smtp Password*	*****
Port Number	25	Enable SSL	<input type="checkbox"/>

Email Content Details

Subject*	Leave Approval	Attachment Folder	
Has Template	<input checked="" type="checkbox"/>	Attachment files	
Content*	<p>Hi {{Name}},</p> <p>Your Leave was approved.</p> <p>Regards,</p> <p>Admin</p>		

Content Editor (Rich Text Area)

Cancel Save

Figure 4-5 – Email configuration Screen

If the notification has the template, it can include a particular template by checking the template check box [refer the image below]. The notification template contains the body of the email with some placeholders.

Template Options

Ruleset Code : Providing a Ruleset Code, the notification manager will choose the template based on the outcome of the rule execution. (i.e.) The Notification manager executes the rule and the result of the rule will be the template name.

More on Template Choosing Rules [here](#)

The notification manager uses this template to send notification. If the rule set is empty, then the notification manager uses the default template.

The template configuration will be discussed in content template page ([here](#)).

Apply Leave Notification

Notification Type: Email

Manage Email Dispatch Details :

Notification Type	Email		
Sender Address*	sender@host.com	Smtp Address*	mail.host.com
Smtp Username*	username	Smtp Password*	*****
Port Number*	25	Enable SSL	<input type="checkbox"/>

Email Content Details

Subject*	Leave Approval	Attachment Folder	
Has Template	<input checked="" type="checkbox"/>	Attachment files	
Template Rule Set	LeaveRule	Template Name*	EMP Template

Clear Cancel Save

Figure 4-6 – Email configuration with template Screen



If the Notification is Global, the tenant admin can override the dispatch and content [there by the content and dispatch will belong to this tenant admin for the global template] details as follows

Manage Email Dispatch Details : Forgot Password Mailer

Notification Type	Email	Cancel	Override Details
Sender Address	user@hostname.com	Smtp Address	localhost
Smtp UserName	user@hostname.com	Smtp Password	*****
Port Number	25	Enable SSL	True
Email Content Details			
Subject	{{{tenantname}}} - Forgot Password	Attachment Folder	
Attachment files			
Template Rule Set	User_PreprocessorRule	Template Name	system notification template

After clicking the Override details, the following screen will appear there by allowing each tenant to configure their own dispatch and content details for the global notification.

Manage Email Dispatch Details : Forgot Password Mailer

Notification Type	Email	Clear	Cancel	Save
Sender Address*	user@hostname.com	Smtp Address*	localhost	
Smtp UserName*	user@hostname.com	Smtp Password*	*****	
Port Number*	25	Enable SSL	<input checked="" type="checkbox"/>	
Email Content Details				
Subject*	Forgot Password	Attachment Folder		
Has Template	<input checked="" type="checkbox"/>	Attachment files		
Template Rule Set	User_PreprocessorRule	Template Name*	system notification template	



Sender Address, SMTP Address, SMTP User Name, SMTP Password and Subject fields are mandatory.

FTP Configuration

On selecting the notification type as FTP, Batch FTP, SFTP or Batch SFTP the following screen will appear.

Document Uploader

Notification Type	Ftp
-------------------	-----

Ftp dispatch details

Ftp Address*	ftp://host.com	Ftp User Name*	ftadmin
Ftp Password*	*****		

Ftp content details

File Path	D://	File Name	document.pdf
-----------	------	-----------	--------------

Cancel Save



Figure 4-7 – FTP Configuration Screen

If the Notification is Global, the tenant admin can override the dispatch and content details as follows

Manage Ftp Details : File upload Notification

Ftp Type: Ftp Ftp Address: ftp://host.com

Ftp User Name: ftpadmin Ftp Password: ***

Add Ftp Content Details

File Path: D:// File Name: employee.pdf

Cancel Override Details

After clicking the Override details, the following screen will appear

Manage Ftp Details : File upload Notification

Ftp Type: Ftp Ftp Address*: ftp://host.com

Ftp User Name*: ftpadmin Ftp Password*: *****

Add Ftp Content Details

File Path: D:// File Name: employee.pdf

Clear Cancel Save



FTPAddress, FTPUserName, and FTPPassword are mandatory.

System Notification Configuration

On selecting notification type as system notification the following screen will appear.

Tenant Request System Notification

Notification Type: SystemNotification

Manage System Notification Details

Notification Type: SystemNotification

Map Id: 4ad211a0-e045-43e9-a088-0f59c

Add System Notification Content

Has Template:

Content:
The tenant (tenantName) wants to add you in that system.

Save Cancel

Figure 4-8 – System Notification Screen

If the system notification has a template we can include the template as follows:



Providing the Ruleset code, the notification manager will choose the template base on the result of the rule execution. (i.e.) The Notification manager executes the rule and the result of the rule execution will be the template name. The notification manager uses this template to send the notification. This is analogous to that of the rule based template definition in the email notifications.

Tenant Request System Notification

Notification Type: SystemNotification

Manage System Notification Details

Notification Type: SystemNotification

Map Id: [Input Field]

Add System Notification Content

Has Template:

System Notification Template Details

Template Rule Set: TenantRequestRule

Template Name*: system notification template

Buttons: Clear, Cancel, Save

Figure 4-9 – System Notification with template Screen

If the Notification is Global, the tenant admin can override the dispatch and content details as follows

Manage System Notification Details : User Approval System Notification

Notification Type: SystemNotification

Map Id

System Notification Template Details

Template Rule Set

Template Name: system notification template

Buttons: Cancel, Override Details (highlighted), Save

After clicking the Override details, the following screen will appear

Manage System Notification Details : User Approval System Notification

Notification Type: SystemNotification

Map Id: [Input Field]

Add System Notification Content

Has Template:

System Notification Template Details

Template Rule Set: [Input Field]

Template Name*: system notification template

Buttons: Clear, Cancel, Save

By using this way, each tenant can configures own dispatch, content details for the global notification.



2. Notification Windows Service

This windows service is responsible for the following activities,

Monitors database at regular intervals and dispatches batch notification (Email / File / System Notification) and audits the same.

Batch Notification Configuration

For batch notifications, the notification windows service needs to install in the machine.

Notifications can be processed as a batch for Email / FTP / System Notifications

Default Batch Parallel thread size is 3. Add the following settings in App.Config appSettings to have different value. Parallel thread size is the number of threads to be created at a time.

```
<add key="ParallelThreadSize" value="2"/>
```

Default Batch Size is 1000. Use the following settings, to change the value. The Batch size is the number of notifications will be taken for the batch processing.

```
<add key="BatchSize" value="500"/>
```

Default Fill Queue Time is 120000ms(2Sec). Use the following settings, to change the value. Fill Queue Time is the time duration between the batch processing.

```
<add key="FillQueueTime" value="180000"/>
```

You want to log the dispatch details then set Log Dispatch Details as True in App.Config appSettings, Then the log file will be created in the Batch Notification Service (window service) installed folder.

```
<add key="LogDispatchDetails" value="true"/>
```

You can see the dispatch details as BatchNotificationDispatchDetails.log inside Notification Service (window service) installed folder.

The config file is available in the Batch Notification Service (window service) installed folder has the above settings with default values.

Notification Manager API

The Notification Manager API is available in CelloSaaS.Notification.ServiceProxies Namespace.

It contains the following methods.

This API is used to send an email or upload a file.

```
/// <summary>
/// This method is used to send an email or a system notification or upload a file.
/// </summary>
/// <param name="notificationId">The notification master id. (Mandatory)</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="notificationContent">The notification Content Details. (Mandatory)</param>
/// <param name="notificationDestination">The notification destination details. (Mandatory)</param>
public static void SendNotification(string notificationId, string tenantId, INotificationContent
notificationContent, INotificationDestination notificationDestination)
```

/* Notification Id, Notification Content and Notification Destination parameters are mandatory */

Sample

Send Email through SendNotification API

```

public void SendNotification_Email()
{
    string notificationId = "3EE5C059-E674-4816-AB71-8CB310D6A14D";
    TextMessageContent notificationContent = new TextMessageContent();
    TextMessageContent.Subject = "Test Mail";
    notificationContent.ProcessedContent = "Hi User,<br> This is Test Mail.";

    EmailDestination notificationDestination = new EmailDestination();
    notificationDestination.SenderAddress = "admin@techcello.com";
    notificationDestination.RecipientAddress = "user@techcello.com";
    notificationDestination.SmtpAddress = "mail.techcello.com";
    notificationDestination.SmtpUserName = "admin@techcello.com";
    notificationDestination.SmtpPassword = "Test";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.SendNotification(notificationId, string.Empty, notificationContent,
    notificationDestination);
}

```

Upload a file through SendNotification API.

```

public void SendNotification_FTP()
{
    string notificationId = "3EE5C059-E674-4816-AB71-8CB310D6A14D";
    FileContent notificationContent = new FileContent();
    notificationContent.FilePath = @"D:\";
    notificationContent.FileName = "TestFile.txt";
    FTPDestination notificationDestination = new FTPDestination();
    notificationDestination.FTPAddress = "FTP:\\" + 127.0.0.1:21";
    notificationDestination.IsSecured = false;
    notificationDestination.UserName = "admin@techcello.com";
    notificationDestination.Password = "test";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.SendNotification(notificationId, string.Empty, notificationContent,
    notificationDestination);
}

```

Send System Notification through SendNotification API

```

public void SendNotification_SystemNotificationTest()
{
    string notificationId = "5ED97DBD-84DD-4BE3-A216-60EBB449AC78";
    SystemNotificationContent notificationContent = new SystemNotificationContent();
    notificationContent.ProcessedContent = "Hi Company Admin,<br> This is Test System Notification";
    SystemNotificationDestination notificationDestination = new SystemNotificationDestination();
    notificationDestination.MapId = "3398f837-b988-4708-999d-d3dfe11875b3";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.SendNotification(notificationId, string.Empty, notificationContent,
    notificationDestination);
}

```

This API reads the email content details from the database using Notification Id and replaces the place holder and dispatches the notification.

/// <summary>

```

/// This reads the content from the configuration using the notificationId, replaces the placeholders
and delivers the notification via notification dispatch manager
/// The kind of content here can only be text content
/// </summary>
/// <param name="notificationId">The notification master id. (Mandatory)</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="placeHolders">The place holders. (Mandatory)</param>
/// <param name="notificationDestination">The notification destination. Mandatory)</param>
public static void ProcessAndSendNotification(string notificationId, string tenantId,
NotificationPlaceHolder placeHolders, INotificationDestination notificationDestination)

```

* Notification id, Place Holder and Notification Destination details are mandatory.

Sample

Send Email through ProcessAndSendNotification API

```

public void ProcessAndSendNotification_Email()
{
    string notificationId = "3EE5C059-E674-4816-AB71-8CB310D6A14D";
    NotificationPlaceHolder notificationPlaceHolder = new NotificationPlaceHolder();
    notificationPlaceHolder.Keys = new Dictionary<string, string>();
    notificationPlaceHolder.Keys.Add("User", "TechCello User");
    EmailDestination notificationDestination = new EmailDestination();
    notificationDestination.SenderAddress = "admin@techcello.com";
    notificationDestination.RecipientAddress = "user@techcello.com";
    notificationDestination.SmtpAddress = "mail.techcello.com";
    notificationDestination.SmtpUserName = "admin@techcello.com";
    notificationDestination.SmtpPassword = "Test";
    notificationDestination.Subject = "Test Mail";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.SendNotification(notificationId, string.Empty, notificationPlaceHolder,
    notificationDestination);
}

```

Send System Notification through ProcessAndSendNotification API

```

public void ProcessAndSendNotification_SystemNotification ()
{
    string notificationId = "5ed97dbd-84dd-4be3-a216-60ebb449ac78";
    string tenantId = "";
    NotificationPlaceholder notificationPlaceHolder = new NotificationPlaceholder();
    notificationPlaceHolder.Keys = new System.Collections.Generic.Dictionary<string, string>();
    NotificationDetails notificationDetails =
    NotificationManagerDetails.GetNotificationDetails(notificationId, "");
    SystemNotificationDestination notificationDestination = new SystemNotificationDestination();
    notificationDestination.MapId = "3398f837-b988-4708-999d-d3dfe11875b3";
    notificationPlaceHolder.Keys.Add("UserName", "jesy");
    notificationPlaceHolder.Keys.Add("TenantName", "john");
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.ProcessAndSendNotification(notificationId, tenantId,
    notificationPlaceHolder, notificationDestination);
}

```

Process the content and send an email or System Notification or upload a file. [FTP has no processing involved]

```
///<summary>
/// This method is used to send an email or upload a file.
/// </summary>
/// <param name="notificationId">The notification master id. (Mandatory)</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="notificationContent"> The notification Content Details. (Mandatory)</param>
/// <param name="notificationDestination">The notification destination Details. (Mandatory)</param>
```

```
public static void ProcessAndSendNotification(string notificationId, string tenantId, INotificationContent notificationContent, INotificationDestination notificationDestination)
```

Sample

To Send Email

```
public void ProcessAndSendNotification_Email ()
{
    string notificationId = "3EE5C059-E674-4816-AB71-8CB310D6A14D";
    TextMessageContent notificationContent = new TextMessageContent();
    notificationContent.Content = "Hi User,<br> This is Test Mail.";
    notificationContent.Subject = "Test Mail";
    EmailDestination notificationDestination = new EmailDestination();
    notificationDestination.SenderAddress = "admin@techcello.com";
    notificationDestination.RecipientAddress = "user@techcello.com";
    notificationDestination.SmtpAddress = "mail.techcello.com";
    notificationDestination.SmtpUserName = "admin@techcello.com";
    notificationDestination.SmtpPassword = "Test";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.ProcessAndSendNotification(notificationId, tenantId, notificationContent,
    notificationDestination);
}
```

To send System Notification

```
public void ProcessAndSendNotification_SystemNotification ()
{
    string notificationId = "89d835dd-2902-401d-9848-e078b3608a82";
    string tenantId = string.Empty;
    SystemNotificationContent notificationContent = new SystemNotificationContent();
    notificationContent.Content = new StringBuilder("Test System Notification Content");
    notificationContent.CreatedBy = notificationContent.UpdatedBy = "Admin";
    SystemNotificationDestination notificationDestination = new SystemNotificationDestination();
    notificationDestination.MapId = "3398f837-b988-4708-999d-d3dfe11875b3";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.ProcessAndSendNotification(notificationId, tenantId, notificationContent,
    notificationDestination);
}
```

To Upload a file

```
public void ProcessAndSendNotification _FTP()
{
    string notificationId = "3EE5C059-E674-4816-AB71-8CB310D6A14D";
    FileContent notificationContent = new FileContent();
    notificationContent.FilePath = @"D:\";
    notificationContent.FileName = "TestFile.txt";
    FTPDestination notificationDestination = new FTPDestination();
    notificationDestination.FTPAddress = @"FTP:\127.0.0.1:21";
    notificationDestination.IsSecured = false;
    notificationDestination.UserName = "admin@techcello.com";
    notificationDestination.Password = "test";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.ProcessAndSendNotification(notificationId, string.Empty,
    notificationContent, notificationDestination);
}
```

Process the content and send batch emails

This method reads the email content details from the data base using Notification Id and replaces the place holder and dispatches the notification.

```
/// <summary>
/// This reads the content from the configuration using the notificationId, replaces the placeholders
/// and delivers the notification via notification dispatch manager
/// The kind of content here can only be batch text content
/// </summary>
/// <param name="notificationId">The notification master id. (Mandatory)</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="placeHolders">The place holders. (Mandatory)</param>
/// <param name="notificationDestination">The notification destination. (Mandatory)</param>
public static void ProcessAndSendBatchNotification(string notificationId, string tenantId,
List<NotificationPlaceHolder> placeHolders, IBatchNotificationDestination notificationDesti
nation)
* Notification id, Place Holders and Notification Destination details are mandatory.
```

Sample

```

public void ProcessAndSendBatchNotification_Email()
{
    string notificationId = "3EE5C059-E674-4816-AB71-8CB310D6A14E";
    string tenantId = string.Empty;
    List<NotificationPlaceHolder> placeHoldersList = new List<NotificationPlaceHolder>();
    NotificationPlaceHolder placeHolders = new NotificationPlaceHolder();
    placeHolders.Keys = new Dictionary<string, string>();
    placeHolders.Keys.Add("UserName", "User1");
    placeHolders.Keys.Add("TenantName", "TechCello");
    placeHoldersList.Add(placeHolders);
    placeHolders = new NotificationPlaceHolder();
    placeHolders.Keys = new Dictionary<string, string>();
    placeHolders.Keys.Add("UserName", "User2");
    placeHolders.Keys.Add("TenantName", "CompanyA");
    placeHoldersList.Add(placeHolders);
    IBatchNotificationDestination notificationDestinationList = new BatchEmailDestination();
    notificationDestinationList.CreatedBy = notificationDestinationList.UpdatedBy = "Admin";
    notificationDestinationList.NotificationDestinations = new List<INotificationDestination>();
    EmailDestination notificationDestination = new EmailDestination();
    notificationDestination.SenderAddress = "admin@techcello.com";
    notificationDestination.RecipientAddress = "user1@techcello.com";
    notificationDestination.SmtpAddress = "mail.techcello.com";
    notificationDestination.SmtpUserName = "admin@techcello.com";
    notificationDestination.SmtpPassword = "Test";
    notificationDestination.Subject = "Test Mail";
    notificationDestinationList.NotificationDestinations.Add(notificationDestination);
    notificationDestination = new EmailDestination();
    notificationDestination.SenderAddress = "admin@techcello.com";
    notificationDestination.RecipientAddress = "user2@techcello.com";
    notificationDestination.SmtpAddress = "mail.techcello.com";
    notificationDestination.SmtpUserName = "admin@techcello.com";
    notificationDestination.SmtpPassword = "Test";
    notificationDestinationList.NotificationDestinations.Add(notificationDestination);
    NotificationServiceProxy.ProcessAndSendBatchNotification(notificationId, tenantId, placeHoldersList,
    notificationDestinationList);
}

```

To send batch emails or batch file upload or upload merged PDF

```

///<summary>
/// Processes and send the batch notification.
///</summary>
///<param name="notificationId">The notification master id.</param>
///<param name="tenantId">The tenant id.</param>
///<param name="content">The notification content details.</param>
///<param name="notificationDestination">The notification destination details.</param>
public static void ProcessAndSendBatchNotification(string notificationId, string tenantId,
IBatchNotificationContent content, IBatchNotificationDestination notificationDestination)
* Notification id, Batch Notification Content Details and Batch Notification Destination details are
mandatory.
IBatchNotificationDestination.NotificationBatchDetails also mandatory.
If notification content details not available in data base then
BatchNotificationDestination.CommonDestinationDetails and
IBatchNotificationContent.CommonContentDetails are also mandatory.
To upload a merged PDF file

```

Set `IBatchNotificationDestination.NotificationBatchDetails.Combine` property as true.

```
IBatchNotificationDestination notificationDestinationList = new BatchEmailDestination();
notificationDestinationList.NotificationBatchDetails = new Notification.Model.NotificationBatch();
notificationDestinationList.NotificationBatchDetails.Combine = true;
```

Add "BatchFolderPath" key and value in App.Config appSetting. This folder will be used to upload the files before merge temporarily.

```
<add key ="BatchFolderPath" value="E:\Projects\CelloSaaS\BatchFileFolder"/>
```

```
public void ProcessAndSendBatchNotificationTest_withoutTenantId_content_email()
{
    string notificationId = "3ee5c059-e674-4816-ab71-8cb310d6a14d";
    string tenantId = string.Empty;
    List<NotificationPlaceHolder> placeHoldersList = new List<NotificationPlaceHolder>();
    BatchTextMessageContent batchNotificationContent = new BatchTextMessageContent();
    batchNotificationContent.NotificationContent = new List<INotificationContent>();
    TextMessageContent textMessageContent = new TextMessageContent();
    textMessageContent.ProcessedContent = "Test Mail";
    textMessageContent.Subject = "Test Mail";
    batchNotificationContent.NotificationContent.Add(textMessageContent);
    textMessageContent = new TextMessageContent();
    batchNotificationContent.Content = new System.Text.StringBuilder("this is content");
    batchNotificationContent.NotificationContent.Add(textMessageContent);
    IBatchNotificationDestination notificationDestinationList = new BatchEmailDestination();
    notificationDestinationList.CreatedBy = notificationDestinationList.UpdatedBy = "Admin";
    notificationDestinationList.NotificationBatchDetails = new Notification.Model.NotificationBatch();
    notificationDestinationList.NotificationBatchDetails.Combine = true;
    notificationDestinationList.NotificationDestinations = new List<INotificationDestination>();
    EmailDestination commonEmailDestination = new EmailDestination();
    commonEmailDestination.SenderAddress = "admin@techcello.com";
    commonEmailDestination.SmtpAddress = "mail.techcello.com";
    commonEmailDestination.SmtpUserName = "admin@techcello.com";
    commonEmailDestination.SmtpPassword = "Test";
    notificationDestinationList.CommonNotificationDestination = commonEmailDestination;
    EmailDestination notificationDestination = new EmailDestination();
    notificationDestination.RecipientAddress = "user2 @techcello.com";
    notificationDestinationList.NotificationDestinations.Add(notificationDestination);
    notificationDestination = new EmailDestination();
    notificationDestination.RecipientAddress = "user2 @techcello.com";
    notificationDestinationList.NotificationDestinations.Add(notificationDestination);
    NotificationServiceProxy.ProcessAndSendBatchNotification(notificationId, tenantId,
    batchNotificationContent, notificationDestinationList);
}
```

To process the content and send emails with this specified locale template
Before using this method, configure the Dispatch and Template details as we mentioned before.

This method reads the email content and template details from the data base using Notification Id and replaces the place holder and delivers the notification.

```

/// <summary>
/// This reads the content from the configuration using the notificationId, replaces
the placeholders
/// based on the locale id and delivers the notification via notification dispatch manager
/// The kind of content here can only be text content
/// </summary>
/// <param name="notificationId">The notification id.</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="localeId">The locale id.</param>
/// <param name="placeholders">The placeholders.</param>
/// <param name="notificationDestination">The notification destination.</param>
public static void ProcessAndSendNotification(string notificationId, string tenantId,
string localeId, NotificationPlaceholder placeholders, INotificationDestination
notificationDestination)
Notification id, localeId, Place Holder and Notification Destination details are mandatory.

```

Sample

```

public void ProcessAndSendNotification_Email_With_LocaleId()
{
    string notificationId = "8a58b676-b9c9-4c9f-905f-6a122ba2c81c";
    string tenantId = "b590cd25-3093-df11-8deb-001ec9dab123";
    NotificationPlaceholder notificationPlaceHolder = new NotificationPlaceholder();
    notificationPlaceHolder.Keys = new System.Collections.Generic.Dictionary<string, string>();
    NotificationDetails notificationDetails =
        NotificationManagerDetails.GetNotificationDetails(notificationId,
        "b590cd25-3093-df11-8deb-001ec9dab123");
    EmailDestination notificationDestination =
        (EmailDestination)notificationDetails.NotificationDestination;
    notificationDestination.RecipientAddress = "user@techcello.com";
    notificationDestination.SecondaryRecipientAddress = new string[] { "user1 @techcello.com",
    "user2 @techcello.com" };
    notificationPlaceHolder.Keys.Add("name", "jesy");
    notificationPlaceHolder.Keys.Add("sender", "john");
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.ProcessAndSendNotification(notificationId, tenantId, "8f5933ba-bf46-48b4-
    b70f-b8d473adb2c6",
    notificationPlaceHolder, notificationDestination);
}

```

To Send System Notification

```

public void ProcessAndSendSystemNotification_With_LocaleId()
{
    string notificationId = "5ED97DBD-84DD-4BE3-A216-60EBB449AC78";
    string tenantId = "";
    NotificationPlaceholder notificationPlaceHolder = new NotificationPlaceholder();
    notificationPlaceHolder.Keys = new System.Collections.Generic.Dictionary<string, string>();
    SystemNotificationDestination notificationDestination = new SystemNotificationDestination();
    notificationDestination.MapId = "3398f837-b988-4708-999d-d3dfe11875b3";
    notificationPlaceHolder.Keys.Add("UserName", "jesy");
    notificationPlaceHolder.Keys.Add("TenantName", "john");
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.ProcessAndSendNotification(notificationId, tenantId, "DA9D2ACB-87D3-
    49A7-9021-0B6B4C9A1ABD",

```

```
notificationPlaceHolder, notificationDestination);
}
```

To send batch emails with the specified locale template

```
/// <summary>
/// Processes the and send batch notification with locale id
/// </summary>
/// <param name="notificationId">The notification id.</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="localeId">The locale id.</param>
/// <param name="placeholders">The placeholders.</param>
/// <param name="notificationDestination">The notification destination.</param>
public static void ProcessAndSendBatchNotification(string notificationId, string tenantId, string
localeId, List<NotificationPlaceholder> placeholders, IBatchNotificationDestination
notificationDestination
Notification id, localeId, Batch Notification Content Details and Batch Notification Destination details
are mandatory.
IBatchNotificationDestination.NotificationBatchDetails also mandatory.
If notification content details not available in data base then
IBatchNotificationDestination.CommonDestinationDetails and
IBatchNotificationContent.CommonContentDetails are also mandatory.
```

Sample

```
public void ProcessAndSendBatchNotification_With_LocaleId()
{
    string notificationId = "09be34c3-0361-42cd-b457-1f9cb0e29e4e";
    string tenantId = "b590cd25-3093-df11-8deb-001ec9dab123";
    List<NotificationPlaceholder> placeHoldersList = new List<NotificationPlaceholder>();
    NotificationPlaceholder placeHolders = new NotificationPlaceholder();
    placeHolders.Keys = new System.Collections.Generic.Dictionary<string, string>();
    placeHolders.Keys.Add("UserName", "ξοην");
    placeHolders.Keys.Add("SenderName", "Εινστειν");
    placeHoldersList.Add(placeHolders);
    placeHolders = new NotificationPlaceholder();
    placeHolders.Keys = new System.Collections.Generic.Dictionary<string, string>();
    placeHolders.Keys.Add("UserName", "αβραηαμ");
    placeHolders.Keys.Add("SenderName", "σμιθ");
    placeHoldersList.Add(placeHolders);
    IBatchNotificationDestination notificationDestinationList = new BatchEmailDestination();
    notificationDestinationList.NotificationDestinations = new List<INotificationDestination>();
    notificationDestinationList.CreatedBy = notificationDestinationList.UpdatedBy = "Admin";
    EmailDestination notificationDestination = new EmailDestination();
    notificationDestination.RecipientAddress = "user@techcello.com";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    notificationDestinationList.NotificationDestinations.Add(notificationDestination);
    notificationDestination = new EmailDestination();
    notificationDestination.RecipientAddress = "user1@techcello.com";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    notificationDestinationList.NotificationDestinations.Add(notificationDestination);
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    NotificationServiceProxy.ProcessAndSendBatchNotification(notificationId, tenantId,
    "8f5933ba-bf46-48b4-b70f-b8d473adb2c6", placeHoldersList, notificationDestinationList);
}
```



```
/// <summary>
/// This method is used to gets the notification details by using the notification id and the tenant id.
/// </summary>
/// <param name="notificationId">The notification id.</param>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
/// <exception cref="NotificationException">If Notification Identifier Not available</exception>
public static NotificationDetails GetNotificationDetails(string notificationId, string tenantId)
```

Sample

```
Public Void GetNotificationDetails()
{
    NotificationDetails notificationDetails = NotificationServiceProxy.GetNotificationDetails("09be34c3-0361-42cd-b457-1f9cb0e29e4e", "b590cd25-3093-df11-8deb-001ec9dab123");
}

/// <summary>
/// This method is used to get the notification details by using the notification id and the tenant id
/// based on the locale id.
/// </summary>
/// <param name="notificationId">The notification id.</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="localeId">The locale id.</param>
/// <returns></returns>
/// <exception cref="CelloSaaS.Notification.NotificationManager.NotificationException">If the
/// notification id is not available</exception>
public static NotificationDetails GetNotificationDetails(string notificationId, string tenantId, string
localeId)
```

Sample

```
public void GetNotificationDetails_With_LocaleId()
{
    NotificationDetails notificationDetails = NotificationServiceProxy.GetNotificationDetails("09be34c3-0361-42cd-b457-1f9cb0e29e4e", "b590cd25-3093-df11-8deb-001ec9dab123", "7b8bc72a-c529-45fb-a83d-7ac708c5f6a2");
}

/// <summary>
/// This method is used to get the notification details by using the notification name and the tenant id.
/// </summary>
/// <param name="notificationName">Name of the notification.</param>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
/// <exception cref="CelloSaaS.Notification.NotificationManager.NotificationException">If the
/// notification id is not available</exception>
public static NotificationDetails GetNotificationDetailsByName(string notificationName, string
tenantId)
```

Sample

```
public void GetNotificationDetailsByName()
{
    NotificationDetails notificationDetails =
    NotificationServiceProxy.GetNotificationDetailsByName("Forgot Password Mailer", " b590cd25-3093-df11-8deb-001ec9dab123");
}
```

```

/// <summary>
/// This method is used to get the notification details by using the notification name with the given
/// locale id.
/// </summary>
/// <param name="notificationName">Name of the notification.</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="localeId">The locale id.</param>
/// <returns></returns>
/// <exception cref="CelloSaaS.Notification.NotificationManager.NotificationException">If the
/// notification id is not available</exception>
public static NotificationDetails GetNotificationDetailsByName(string notificationName, string
tenantId, string localeId)

```

Sample

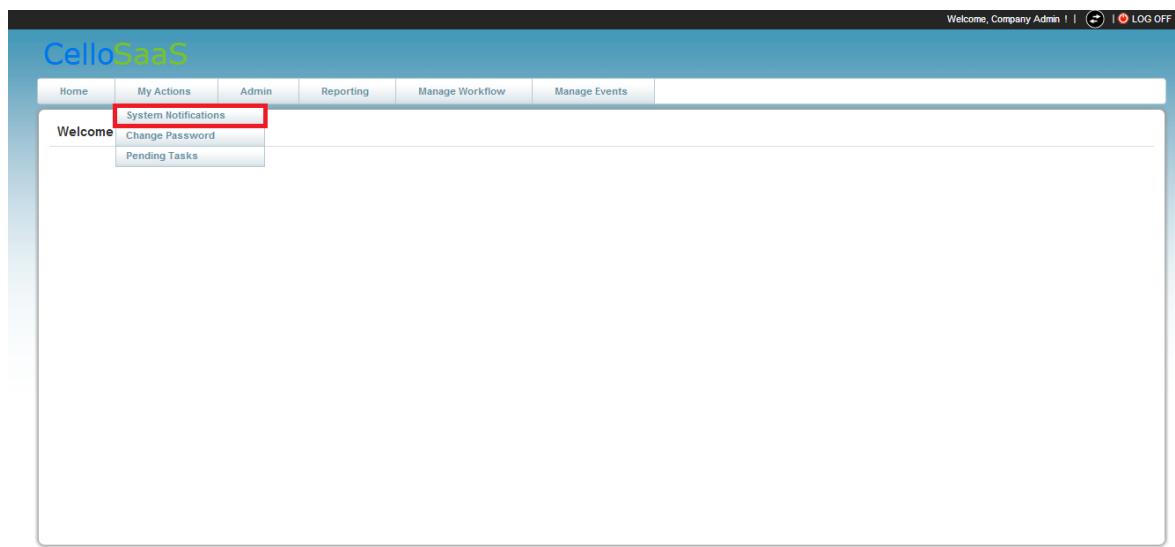
```

public void GetNotificationDetailsByNameAndLocaleId()
{
    NotificationDetails notificationDetails =
    NotificationServiceProxy.GetNotificationDetailsByName("Forgot Password Mailer", " b590cd25-
3093-df11-8deb-001ec9dab123", "a1d85bdf-986a-427d-961f-6ff7df839c53");
}

```

4.8 System Notifications

The System Notifications, received by the logged in user can viewed as follows.



The logged in user's system notification details appears as displayed as shown below [Refer Screen No. 4-12]. System Notifications based on particular date and time can be filtered using the search box.

The MapId is used to identify the receiver of the system notification. Cello sets this property as the User Id, however the application can override to use like an Employee Id or so.

Notification Name	Dispatched Time	Content	Map Id	Notification Status
User Approval System Notification	11/02/13 5:49 PM	Test Template for System Notification.	3398fb37-b988-4708-999d-d3df11875b3	Success
User Approval System Notification	11/02/13 5:48 PM	Test Template for System Notification.	3398fb37-b988-4708-999d-d3df11875b3	Success

Figure 4-2 – System Notification Screen

Notification Audit Details

Once an email/batch Emails are sent or a file/batch files is uploaded or the system notification/batch system notifications are sent the dispatch details (Audit Details) can be viewed by tenant administrator as follows.

Figure 4-3 – Notification Audit Menu

Notification Audit details

Email Audit Details

Search can be performed based on Notification Type, such as Email or Batch Email along with the dispatched time of notification.

Notification Audit Details

Tenant: Company

Notification type *	<input type="text" value="Email"/>	Notification name	<input type="text" value="-All-"/>	Status	<input type="text" value="-All-"/>
Start time	<input type="text"/>	End time	<input type="text"/>	<input type="button" value="Search"/>	

Notification Name	Created on	Sender address	Recipient address	Content	Notification status
Email	06/02/13 11:08 AM	venkat.ramasamy@aspiresys.com	venkat.ramasamy@aspiresys.com	test	Failure
Email	01/02/13 5:13 PM	venkat.ramasamy@aspiresys.com	venkat.ramasamy@aspiresys.com	Test Email	Success

Show 10 entries Showing 1 to 1 of 1 entries

Figure 4-44 –Email Audit Screen

FTP Audit Details

Search can be performed based on Notification Type such as FTP or Batch FTP, along with the dispatched time of notification.

Notification Audit Details

Tenant: Company

Notification type *	<input type="text" value="Ftp"/>	Notification name	<input type="text" value="-All-"/>	Status	<input type="text" value="-All-"/>
Start time	<input type="text"/>	End time	<input type="text"/>	<input type="button" value="Search"/>	

Notification Name	Created on	Ftp address	File path	File name	Notification status
File upload Notification	08/02/13 1:00 PM	ftp://127.0.0.1/	D://	XmSerializer.cs	Failure
File upload Notification	08/02/13 12:59 PM	ftp://127.0.0.1/	D://	XmSerializer.cs	Success
File upload Notification	08/02/13 12:57 PM	ftp://127.0.0.1/	D://	XmSerializer.cs	Success

Show 10 entries Showing 1 to 1 of 1 entries

Figure 4-55 –FTP Audit Screen

System Notification Audit Details

Search can be performed based on Notification Type such as System Notification or Batch System Notification, along with the dispatched time of notification



Notification Audit Details

Tenant: Company

Notification Name	Dispatched Time	Content	Map Id	Notification Status
User Approval System Notification	11/02/13 5:49 PM	Test Template for System Notification.	3398f837-b988-4708-999d-d3dfe11875b3	Success
User Approval System Notification	11/02/13 5:48 PM	Test Template for System Notification.	3398f837-b988-4708-999d-d3dfe11875b3	Success

Show 10 entries Showing 1 to 1 of 1 entries

Figure 4-16 – System Notifications Audit Details Screen

```
To get the Notification Audit Details
/// <summary>
/// Gets the notification audit details.
/// </summary>
/// <param name="notificationAuditSearchCondition">The notification audit search
condition.</param>
/// <returns></returns>
public static List<NotificationAudit> GetNotificationAuditDetails (NotificationAuditSearchCondition
notificationAuditSearchCondition)
```

4.9 Template Management

These Templates are tenant specific templates and used in Notifications or Events. The content of template can be drafted for different locales.

To Configure a Template

Welcome

Manage Notifications

Content Template

Figure 4-6 – Content Template Menu

Content Template List

The screenshot shows a 'Template Management' section within the CelloSaaS interface. At the top right is a green 'Add' button. Below it is a search bar with a magnifying glass icon and a 'GO' button. A table lists six entries:

Template name	Description	Category	Manage Template	Edit
Add Tenant Template	Add Tenant Template	EventTemplate		
Add User Template	User Creation Template	EventTemplate		
Approve Leave Template		NotificationTemplate		
Deactivate User Template	User Deactivation Template	EventTemplate		
Login Password Failure Template	Login Password Failure Template	EventTemplate		
Tenant Approval Template		NotificationTemplate		

Below the table, a message says 'Show 10 entries Showing 1 to 6 of 6 entries'. Navigation arrows are at the bottom right.

Figure 4-7 - Template Management Screen

To add a template [Notification/Event]

The screenshot shows a 'Manage Template' dialog box. At the top right are 'Cancel' and 'Save' buttons. The form fields are:

- Template name*:
- Template description:
- Template category:

Figure 4-8 – Manage Template Screen

To add Locale based template content

The screenshot shows the 'Manage Templates : Tenant Approval Template' interface. At the top, there's a search bar and a toolbar with 'Back' and 'Add' buttons. Below the search bar is a table titled 'Template Content' with one entry: 'Hi {{Name}},

The Tenant {{tenant...}}'. To the right of the table are 'Edit Details' and 'Delete Details' buttons. Below the table is a large 'Add Template Details' section. It has a 'Locale Name' dropdown set to 'en'. A rich text editor contains the placeholder 'Hi {{Name}}'. Below the editor, the text 'The Tenant {{tenant}} was approved the user.' and 'Regards,
Tenant Admin.' is visible. At the bottom right of this section are 'Cancel' and 'Save' buttons.

Figure 4-9 - Add Template Details Screen

For the locale based content templates, two types of placeholders are supported

1. Entity type Placeholders.
2. Xml type Placeholder.

For entity type, it requires the objects to replace the Placeholders. The placeholder should be as follows.

Example

```
{UserDetails.MembershipDetails.UserName}
```

Here, the placeholder requires the UserDetails object to replace the placeholder.

For xml type, it requires xml to replace the Placeholder. The placeholder should be as follows.

Example

```
{Element("/UserDetails/MembershipDetails/UserName")}
```

Here the placeholder requires the xml to replace the placeholder. The xml can be set by the following way.

Example

```
String xml =
"<Root><UserDetails><MembershipDetails><UserName>John</UserName></MembershipDetails>
</UserDetails></Root>";
```

```
PlaceholderXml placeholderXml = new PlaceholderXml{ Xml = xml};
```



Then this placeholderXml object will be passed through the notification placeholder while sending the notification.

Locale Management

The screenshot shows the CelloSaaS Admin interface. The top navigation bar includes links for Home, My Actions, Admin, Reporting, Manage Workflow, and Manage Events. The Admin section is expanded, showing sub-links for Tenant Management, User Management, Access Management, Configuration Management, Manage Notifications, Audits, Refresh Cache, Manage Entities, Manage Data View, Manage Settings Templates, Module Configuration, Manage Business Rules, Manage PickupLists, PickupList Relationships, Manage MasterData, and Locale Names. The 'Locale Names' link is highlighted with a red box.

Figure 4-21 - Locale Names Menu

Locale Names Details

The screenshot shows the 'Manage LocaleNames Details' page. The top navigation bar is identical to Figure 4-21. The main content area has a search bar with a 'GO' button and a table titled 'Manage LocaleNames Details'. The table has columns for 'Locale Name', 'Description', 'Manage', and 'Delete'. It contains two entries: 'en' with a description of 'General English' and 'Manage' and 'Delete' buttons; and 'hi' with a description of 'Hindi' and 'Manage' and 'Delete' buttons. Below the table is a pagination control showing 'Show 10 entries Showing 1 to 2 of 2 entries'.

Locale Name	Description	Manage	Delete
en	General English	[Manage]	[Delete]
hi	Hindi	[Manage]	[Delete]

Figure 4-22 - Locale Names Details



Add Locale

Locale Name	Description
en	General English

Figure 4-23 – Add LocaleId Screen

The Locale Id will be added by using the master Data configuration, after adding the locale Id, we can use it for configuring the notification templates.

Step by step guide to use notification features:

Scenario 1:

For example, let us take Leave approval process notification

1. UI level configuration:

Create notification master called “Leave Approval Notification”.

Click manage dispatch icon for “Leave Approval Notification”.

Select the notification type as email and fill the appropriate values.

If the notification has template, then create a template and add the locale based content for the template. The locale “en” is the default locale.

Now the notification is ready to use.

2. Send the “Leave Approval Notification” using the API

The configured notification is dispatched as follows

```
public void SendNotification ()  
{  
    string notificationId = "9a1f03cf-730b-43a4-8498-91c04ad87a64";  
    string tenantId = "6e98443f-4724-e211-b91a-7845c4058d3d";  
  
    NotificationDetails notificationDetails = NotificationServiceProxy .GetNotificationDetails  
(notificationId, tenantId);  
  
    EmailDestination notificationDestination = (EmailDestination)  
    notificationDetails.NotificationDestination;  
  
    NotificationPlaceholder notificationPlaceholder = new NotificationPlaceholder();  
    string inputXml = "<root><country>Us</country> <state>Texas</state> <city>Fortville</city>  
<Name>John</Name></root>";
```

```
PlaceholderXml placeholderXml = new PlaceholderXml(inputXml);
notificationPlaceholder.ObjectPlaceholders = new object[] { placeholderXml };
notificationPlaceholder.Keys = new System.Collections.Generic.Dictionary<string, string>();
notificationPlaceholder.Keys.Add("Admin", "Smith");
notificationDestination.RecipientAddress = "user@host.com";
notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
NotificationServiceProxy.ProcessAndSendNotification(notificationId, tenantId,
notificationPlaceholder, notificationDestination);
}
```

The Notification id will be in the NotificationMaster Table.

Scenario 2:

For example, let us take Document upload process notification

1. UI level configuration:

Create notification master called “Document Uploads Notification”.
Click manage dispatch icon for “Document Uploads Notification”.
Select the notification type as FTP and fill the appropriate values.

Now the notification is ready to use.

2. Send the “Leave Approval Notification” using the API

The configured notification is dispatched as follows

```
public void SendNotification ()
{
    string notificationId = "ca1f03cf-730b-43a4-8498-91c04ad87a64";
    string tenantId = "6e98443f-4724-e211-b91a-7845c4058d3d";

    NotificationDetails notificationDetails = NotificationServiceProxy.GetNotificationDetails
(notificationId, tenantId);

    FTPDestination notificationDestination = (FTPDestination) notificationDetails.NotificationDestination;

    FileContent notificationContent = new FileContent();
    notificationContent.FilePath = @"D:\FTPSource";
    notificationContent.FileName = "Resume.pdf";
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";

    NotificationServiceProxy.ProcessAndSendNotification (notificationId, tenantId, notificationContent,
notificationDestination);
}
```

Scenario 3:

For example, let us take Tenant request process notification

1. UI level configuration:

Create notification master called “Tenant Request Notification”.
Click manage dispatch icon for “Tenant Request Notification”.
Select the notification type as System Notification and fill the appropriate values.
If the notification has template, then create a template and add the locale based content for the template. The locale “en” is the default locale.



Now the notification is ready to use.

2. Send that “Leave Approval Notification” using the API

The configured notification is dispatched as follows

```
public void SendNotification ()  
{  
    string notificationId = "7a1f03cf-730b-43a4-8498-91c04ad87a64";  
    string tenantId = "6e98443f-4724-e211-b91a-7845c4058d3d";  
  
    NotificationDetails notificationDetails = NotificationServiceProxy.GetNotificationDetails  
(notificationId, tenantId);  
  
    SystemNotificationDestination notificationDestination = new SystemNotificationDestination();  
    notificationDestination.MapId = "3398f837-b988-4708-999d-d3dfe11875b3";  
  
    SystemNotificationContent notificationContent =  
(SystemNotificationContent)notificationDetails.NotificationContent;  
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";  
  
    NotificationServiceProxy.ProcessAndSendNotification (notificationId, tenantId, notificationContent,  
    notificationDestination);  
}
```

Scenario 4:

For example, let us take Career Development Process notification

1. UI level configuration:

Create notification master called “Career Development Process Notification”.

Click manage dispatch icon for “Career Development Process Notification”.

Select the notification type as Batch Email and fill the appropriate values.

If the notification has template, then create a template and add the locale based content for the template. The locale “en” is the default locale.

Now the notification is ready to use.

2. Send the “Career Development Process Notification” using the API

The configured notification is dispatched as follows.

```
public void SendBatchNotification()  
{  
    string notificationId = "4BF24E52-A243-4229-8FCC-26027C0A1153";  
    string tenantId = "6E98443F-4724-E211-B91A-7845C4058D3D";  
    List<NotificationPlaceholder> placeHoldersList = new List<NotificationPlaceholder>();  
    NotificationPlaceholder placeHolders = new NotificationPlaceholder();  
    string inputXml = "<root><country>Us</country> <state>Texas</state> <city>Fortville</city>  
<Name>John</Name></root>";  
    PlaceholderXml placeholderXml = new PlaceholderXml(inputXml);  
  
    placeHolders.ObjectPlaceholders = new object[] { placeholderXml };  
    placeHolders.Keys = new System.Collections.Generic.Dictionary<string, string>();  
    placeHolders.Keys.Add("UserName", "Smith");  
    placeHolders.Keys.Add("SenderName", "John");  
    placeHoldersList.Add(placeHolders);
```

```

placeHolders = new NotificationPlaceholder();
placeHolders.Keys = new System.Collections.Generic.Dictionary<string, string>();
placeHolders.Keys.Add("UserName", "Smith");
placeHolders.Keys.Add("SenderName", "Caroline");
inputXml = "<root><country>Us</country> <state>Texas</state> <city>Fortville</city>
<Name>John</Name></root>";
placeholderXml = new PlaceholderXml(inputXml);
placeHolders.ObjectPlaceholders = new object[] { placeholderXml };
placeHoldersList.Add(placeHolders);

IBatchNotificationDestination notificationDestinationList = new BatchEmailDestination();
notificationDestinationList.NotificationDestinations = new List<INotificationDestination>();
notificationDestinationList.CreatedBy = notificationDestinationList.UpdatedBy = "Admin";
NotificationDetails notificationDetails = NotificationServiceProxy
    .GetNotificationDetails(notificationId, tenantId);

BatchEmailDestination BnotificationDestination = (BatchEmailDestination)
notificationDetails.NotificationDestination;
EmailDestination notificationDestination = (EmailDestination)
BnotificationDestination.CommonNotificationDestination;
notificationDestination.RecipientAddress = "receiver1@host.com";
notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
notificationDestinationList.NotificationDestinations.Add(notificationDestination);

notificationDestination = (EmailDestination) BnotificationDestination.CommonNotificationDestination;
notificationDestination.RecipientAddress = "receiver2@host.com";
notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
notificationDestinationList.NotificationDestinations.Add(notificationDestination);
notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";

NotificationServiceProxy.ProcessAndSendBatchNotification(notificationId, tenantId,
placeHoldersList, notificationDestinationList);
}

```

Note:

While sending the batch notifications, make sure that the notification windows service is running.

Scenario 5:

Bulk Document Upload notification sample

1. UI level configuration:

Create notification master called “Bulk Document Upload Notification”.
Click manage dispatch icon for “Bulk Document Upload Notification”.
Select the notification type as Batch FTP and fill the appropriate values.

Now the notification is ready to use.

2. Send that “Bulk Document Upload Notification” using the API

The configured notification is dispatched as follows.

```

public void SendBatchNotification()
{
    string notificationId = "a5ee7dce-1cb9-4e6d-ae63-cbeadf5cb618";

```

```
string tenantId = "6E98443F-4724-E211-B91A-7845C4058D3D";

BatchFileContent batchNotificationContent = new BatchFileContent();
batchNotificationContent.NotificationContent = new List<INotificationContent>();
FileContent filecontent = new FileContent();
filecontent.FilePath = @"C:\";
filecontent.FileName = "File.pdf";
batchNotificationContent.NotificationContent.Add(filecontent);

FileContent empFilecontent = new FileContent();
empFilecontent.FilePath = @"C:\";
empFilecontent.FileName = "Resume.pdf";
batchNotificationContent.NotificationContent.Add(empFilecontent);
batchNotificationContent.CommonNotificationContent = empFilecontent;

BatchFTPDestination notificationDestinationList = new BatchFTPDestination();
notificationDestinationList.NotificationBatchDetails = new Notification.Model.NotificationBatch();
notificationDestinationList.NotificationBatchDetails.Combine = false;
notificationDestinationList.NotificationDestinations = new List<INotificationDestination>();
notificationDestinationList.CreatedBy = notificationDestinationList.UpdatedBy = "Admin";

FTPDestination notificationDestination = new FTPDestination();
notificationDestinationList.FTPAddress = notificationDestination.FTPAddress = "FTP://127.0.0.1:21";
notificationDestination.UserName = "admin";
notificationDestination.Password = "password";
notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
notificationDestinationList.CommonNotificationDestination = notificationDestination;
notificationDestinationList.NotificationDestinations.Add(notificationDestination);

FTPDestination destination = new FTPDestination();
destination.FTPAddress = "FTP://198.0.0.1:21";
destination.UserName = "user";
destination.Password = "password";
destination.CreatedBy = destination.UpdatedBy = "Admin";
notificationDestinationList.NotificationDestinations.Add(destination);

NotificationServiceProxy.ProcessAndSendBatchNotification(notificationId, tenantId,
batchNotificationContent, notificationDestinationList);
}
```

Here , the configured Notification content and destination details can be used as the common destination or content details.

Scenario 6:
Bulk System Notification Sample

1. UI level configuration:

Create notification master called “Bulk System Notification”.
Click manage dispatch icon for “Bulk System Notification”.
Select the notification type as Batch System Notification and fill the appropriate values.
If the notification has template, then create a template and add the locale based content for the template. The locale “en” is the default locale.

Now the notification is ready to use.

2. Send the “Leave Approval Notification” using the API

The configured notification is dispatched as follows

```
public void SendBatchSystemNotification()
{
    string notificationId = "10d4cf99-fd23-401d-a7b5-134627c72661";
    string tenantId = "c0d4cf99-fd23-401d-a7b5-134627c72661";
    List<NotificationPlaceholder> placeHoldersList = new List<NotificationPlaceholder>();

    NotificationPlaceholder placeHolders = new NotificationPlaceholder();
    placeHolders.Keys = new System.Collections.Generic.Dictionary<string, string>();
    placeHolders.Keys.Add("UserName", "Smith");
    placeHolders.Keys.Add("TenantName", "techcello");
    placeHoldersList.Add(placeHolders);

    placeHolders = new NotificationPlaceholder();
    placeHolders.Keys = new System.Collections.Generic.Dictionary<string, string>();
    placeHolders.Keys.Add("UserName", "John");
    placeHolders.Keys.Add("TenantName", "techcello");
    placeHoldersList.Add(placeHolders);

    NotificationDetails notificationDetails =
    NotificationManagerDetails.GetNotificationDetails(notificationId, tenantId);

    IBatchNotificationDestination notificationDestinationList = new
    BatchSystemNotificationDestination();
    notificationDestinationList.NotificationDestinations = new List<INotificationDestination>();
    notificationDestinationList.CreatedBy = notificationDestinationList.UpdatedBy = "Admin";
    BatchSystemNotificationDestination BnotificationDestination =
    (BatchSystemNotificationDestination)notificationDetails.NotificationDestination;
    SystemNotificationDestination notificationDestination =
    (SystemNotificationDestination)BnotificationDestination.CommonNotificationDestination;
    notificationDestination.CreatedBy = notificationDestination.UpdatedBy = "Admin";
    notificationDestinationList.NotificationDestinations.Add(notificationDestination);

    notificationDestinationList.NotificationDestinations.Add(notificationDestination);
    NotificationManagerDetails.ProcessAndSendBatchNotification(notificationId, tenantId,
    placeHoldersList, notificationDestinationList);
}
```

How to Guide: [Notification Management](#)

4.10 Tenant Encapsulation

It is extremely important to keep a tenant related data separated from each other so that no tenant gets access to data of a different tenant. CelloSaaS framework handles this carefully by isolation tenant data from others.

While CelloSaaS supports different data isolation mechanisms, this tenant encapsulation mechanism helps in protecting the developers from not making any mistakes even unknowingly while writing queries. Developers do not need to refer to a tenant IDs in their queries by using this mechanism. This helps in data security and also helps in developer productivity.

An user is always linked to a tenant, and the requests from a logged on user must automatically retrieve data of the respective tenant only. Except product admin, it should still be possible to get access data of tenants from CelloSaaS database by writing direct queries without tenant ID.

Steps to implement Tenant Encapsulation



Your tenant related tables should have a column named as "TenantId". You need to use CelloSaaS DBAccess class to perform all operations. Add "AutolsolationTenant" key as TRUE in appsetting tag in web.config file.

Following steps need to be followed while writing query in DAL layer.

Insert Query

Set the PrimaryTableName parameter in the insert query, failing it will throw an error message.
Insert query should contain columns and values.

Insert query class contains three properties

```
classInsertQuery : TransactionalQuery
{
    ///<summary>
    /// Give the columns need to insert to the table.
    /// Columns can be with in parenthesis with Comma separated.
    ///</summary>
    publicStringBuilder Columns { get; set; }
    ///<summary>
    ///Provide full insert query for the table.
    ///</summary>
    [Obsolete]
    publicStringBuilder QueryWithCondition { get; set; }
    ///<summary>
    /// Values for the columns specified in the same order with comma separated.
    /// Values can be with in parenthesis.
    ///</summary>
    publicStringBuilder Values { get; set; }
}
```

There are two ways of writing insert query.

Provide Columns and Values to the Insert query.

Add columns to the column property. Do not specify TenantId column in column,CelloSaaS will take care of it.

```
Columns.Append((CustomerInfo_Name, CustomerInfo_Description,
CustomerInfo_CompanyName, CustomerInfo_CustomerRevenue, CustomerInfo_NoOfEmployees,
CustomerInfo_Vertical, CustomerInfo_Status, and CustomerInfo_SaleExecutive));
```

Add values to the values property of insert query.

```
values.Append("@name, @description, @companyName, @revenue, @noOfEmployees, @vertical, @status, @saleExecutive")
```

Provide full insert query

You can provide full insert query(i.e. insert into columns and values) in QueryWithCondition parameter.

```
StringBuilder query = new StringBuilder();
query.Append("Insert into CRM.LeadDetails(Lead_Name,Lead_AddressId, Lead_LeadStatus,
Lead_CreatedOn, Lead_CreatedBy, Lead_Status ) OUTPUT Inserted.Lead_ID ");
query.Append(" values (@name, @addressId, @leadStatus, @createdOn, @createdBy,
@status);");
InsertQuery insertQuery = new InsertQuery(getConnectionStringName());
insertQuery.QueryWithCondition = query;
```

Add DbParameter to the insert query. Use DBAccess class and call Execute methods to execute the query.

```
BAccess dbAccess = new DBAccess(getConnectionStringName());
String Identifier = dbAccess.InsertAndReturnGUID(insertQuery, "Lead_ID");
```

Add DbParameter to the insert query. Use DBAccess class and call Execute methods to execute the query.

```
DBAccess dbAccess = new DBAccess(getConnectionStringName());
String Identifier = dbAccess.InsertAndReturnGUID(insertQuery, "Lead_ID");
```

Update Query

Set the PrimaryTableName parameter in update query.

```
updateQuery.PrimaryTableName = "CustomerInfo";
CelloSaaS will throw an error message if the PrimaryTableName property is not set.
```

Update query class contains three properties

```
public class UpdateQuery : TransactionalQuery
{
    ///<summary>
    ///Provide the condition for the update query.
    ///</summary>
    public String Builder Condition { get; set; }
    ///<summary>
    /// Provide the query for the update with out condition.
    ///</summary>
    public String Builder Query { get; set; }
    ///<summary>
    /// Provide the full update query.
    ///</summary>
    [Obsolete]
    public String Builder QueryWithCondition { get; set; }
}
```

Two ways of providing update query.

Provide Query and Condition to the update query.

Set query without condition to query property in update query.

```
StringBuilder query = new StringBuilder();
query.Append("UPDATE CustomerInfo SET CustomerInfo_Name =
@name,CustomerInfo_Description = @description,CustomerInfo_CompanyName =
@companyName,CustomerInfo_CustomerRevenue=
@revenue,CustomerInfo_NoOfEmployees=@noOfEmployees,CustomerInfo_Vertical=@vertical,Cus
tomerInfo_SaleExecutive=@saleExecutive");
```

Set condition for update query to condition property in UpdateQuery.

```
StringBuilder condition = newStringBuilder();
condition.Append("CustomerInfo_Id = @id");
```

Provide full update query

You can provide full update query (i.e. query and condition) in QueryWithConditionparameter.

```
StringBuilder query = newStringBuilder();
query.Append("Update CRM.LeadDetails set Lead_Name = @name,Lead_LeadStatus =
@leadStatus,Lead_UpdatedBy= @updatedBy,Lead_UpdatedOn = @updatedOn where
Lead_Id= @id ");
UpdateQuery updateQuery = newUpdateQuery(getConnectionStringName());
updateQuery.QueryWithCondition = query;
```

Add the DbParameter to the update query. Use DBAccess class and call the Execute methods to execute the query.

```
DBAccess dbAccess = newDBAccess(getConnectionStringName());
int id = dbAccess.ExecuteNonQuery(updateQuery);
```

Delete Query

Set the PrimaryTableName parameter in delete query.

```
DeleteQuery deleteQuery = newDeleteQuery(getConnectionStringName());
deleteQuery.PrimaryTableName = "CustomerInfo";
```

Delete query class contains three properties.

```
public class DeleteQuery : TransactionalQuery
{
    ///<summary>
    ///Provide the condition for the update query.
    ///</summary>
    public StringBuilder Condition { get; set; }
    ///<summary>
    /// Provide the query for the update with out condition.
    ///</summary>
    public StringBuilder Query { get; set; }
    ///<summary>
    /// Provide the full update query.
    ///</summary>
    [Obsolete]
    public StringBuilder QueryWithCondition { get; set; }
}
```

There are two ways of providing delete query.

Provide Query and Condition to the delete query.

Set the query without condition to the query property in delete query.

```
StringBuilder query = newStringBuilder();
query.Append("Delete CustomerInfo");
```

Set the condition for the delete query in condition property in delete query.

```
StringBuilder condition = newStringBuilder();
condition.Append("CustomerInfo_Id = @Id");
```

Provide full update query

You can provide full delete query (i.e. query and condition) in QueryWithConditionparameter.

```
StringBuilder query = newStringBuilder();
query.Append("Delete CRM.CustomerInfo where CustomerInfo_Id =@id");
DeleteQuery deleteQuery = newDeleteQuery(getConnectionStringName());
deleteQuery.QueryWithCondition = query;
```

Add the DbParameter to the update query. Use DBAccess class and call the Execute methods to execute the query.

```
DBAccess dbAccess = newDBAccess(getConnectionStringName());
int returnValue = dbAccess.ExecuteNonQuery(deleteQuery);
```

Fetch Query

Use the CelloSaaS tools to create a view for the tenant specific tables. cellosaas tools will get serve name, database name, username and password for the database and generate the view for the tables which have column name “TenantId”.



Set the PrimaryTableName parameter in fetch query. Use view names instead of table name for tenant specific tables.

4.11 Automation of Data Access Layer

Data access automation service exposes a well defined API that can be used to create, update, fetch, fetch list and delete entities without writing implementation for these operations.

How to Use DataAccessAutomationService

1. Create Data Base Tables
2. Create Entity Model with Mapping Attributes

Model should inherit from TenantTransactionalEntity.

All properties should be virtual. Should contain get and set properties.

Add the following section in your web.config file.

```
<section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler,
NHibernate" />
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
<session-factory>
<property
name="connection.provider">CelloSaaS.Library.Provider.NHibernateConnectionProvider,
CelloSaaS.Library</property>
<property name="proxyfactory.factory_class">NHibernate.ByteCode.LinFu.ProxyFactoryFactory,
NHibernate.ByteCode.LinFu</property>
<property name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
<property name="connection.connection_string">Data Source=(local);Initial
Catalog=databaseName;Integrated Security=True</property>
<property name="dialect">NHibernate.Dialect.MsSql2000Dialect</property>
<property name="show_sql">false</property>
</session-factory>
</hibernate-configuration>
```

Create a new Mapping Folder in your MVC Application Directory.

Add Mapping key in App Settings with your mapping folder path.

```
<add key="Mapping" value="E:\SaaSMVCApplication\MappingFiles" />
```

Add the following dll references in your MVC Application (shipped with CelloSaaS)

```
NHibernate.dll  
LinFu.DynamicProxy.dll  
NHibernate.ByteCode.LinFu.dll  
lesi.Collections.dll
```

Call the DataAccessAutomationService API's to create/fetch/search/update/delete entities from the database.

Add Hibernate Model dll file path in app config with DataAccessAutomationAssbly as key.

```
<add key="DataAccessAutomationAssembly" value="Test.Model.dll, Test1.Model.dll"/>
```



For First time use, the API will create the mapping xml files and store them in the mapping folder defined in the App key for later use.

Supported Relationships

One-To-One

Two entities having same identifiers (primary keys) then you can use One-To-One relationship.

One-To-Many

A One-To-Many association links the tables of two classes via a foreign key, with no intervening collection table. Currently cellosaas supports only ICollection<T> type.

Many-To-One

A foreign key in one table refers the primary key column(s) of the target table. This is the simplest parent/child relationship.

Mapping Attributes

EntityAttribute

Entity attribute contains the following and is used to decorate your model class.

Entity Attribute	Parameter	Description
Name	Optional	The entity name for the model. Defaults to unqualified class name.
TableName	Mandatory	The name of its database table.
Lazy	Optional	Enables or disables lazy loading. Defaults to true. You can disable it by setting it to false.
IdentifierColumnName	Mandatory	The primary key column name of the database table.
IdentifierType	Mandatory	The type of the primary key column of the database. See IdentifierDataType.

```
[Entity("tableName", IdentifierDataType.Int, "PrimaryKeyColumnName")]
class Model : TransactionalBaseEntity{
// data members
}Example
```

SampleAttribute

This attribute is used to decorate simple properties of your model and it contains the following:

ColumnName

NotNull

TypeName

Length

Insert

Update

Entity Attribute	Parameter	Description
ColumnName	Optional	The column name of the database table. Defaults to property name.
NotNull	Optional	Defaults to false. If true the property value cannot contain null value.
TypeName	Optional	A name that indicates the property type.
Length	Optional	If the property type is string, then you can specify maximum length of the string value.
Insert	Optional	If true the property will be included while performing insert operation on the database table. If false the property will not be included while performing insert operation on the database table. Defaults to true.
Update	Optional	If true the property will be included while performing update operation on the database table else not included. Defaults to true. For Example: [Simple] public virtual string Name { get; set; }

OneToOneAttribute

A one to one relational model can be defined using this attribute.

Constrained

ReferenceEntityName

Entity Attribute	Parameter	Description
Constrained	Optional	Specifies that a foreign key constraint on the primary key of the mapped table references the table of the associated class. Defaults to false.
ReferenceEntity Name	Optional	Name of the entity it refers to. Defaults to the class type of the property.

		<p>For Example:</p> <pre>[OneToOne(Constrained=true)] public virtual ContactDetails Contact { get; set; }</pre>
TypeName	Optional	A name that indicates the property type.
Length	Optional	If the property type is string, then you can specify maximum length of the string value.
Insert	Optional	If true the property will be included while performing insert operation on the database table. If false the property will not be included while performing insert operation on the database table. Defaults to true.
Update	Optional	If true the property will be included while performing update operation on the database table else not included. Defaults to true. For Example: <pre>[Simple] public virtual string Name { get; set; }</pre>

OneToManyAttribute

If the relational model is a one-to-many association then this attribute can be used to define such relations in your model.

TableName

ReferenceEntityName

Lazy

Inverse

KeyColumnName

KeyColumnNotNull

IndexColumnName

IndexColumnType

Access

Cascade

Entity Attribute	Parameter	Description
TableName	Mandatory	Name of the database table it relates to.
ReferenceEntityName	Optional	Name of the entity it refers to. Defaults to the class type of the property.
Lazy	Optional	Enables or disables lazy loading. Defaults to true.
Inverse	Optional	If false then it will update the child relation with null values while insert/update/delete the parent record. Defaults to true.
KeyColumnName	Mandatory	Primary key column name of the table.

KeyColumnNotNull	Optional	Specifies whether it can contain null value or not. Defaults to true.
IndexColumnName	Mandatory if using list	Currently not supported.
IndexColumnType	Optional	Currently not supported.
Access	Optional	Specifies how the data from database is injected into your model. Defaults to AccessType.Property. See AccessType.
Cascade	Optional	Specifies how the child records are affected during CURD operations. Defaults to CascadeType.All. See CascadeType. For Example: [OneToMany("RoleTbl", "RoleId", Cascade=CascadeType.SaveUpdate)] public virtual ICollection<Roles> roles { get; set; }

Currently ICollection<T> type has to be used to define such relationships.

ManyToOneAttribute

An ordinary association to another persistent class is declared using this attribute. The relational model is a many-to-one association. It's just an object reference.

Entity Attribute	Parameter	Description
ColumnName	Mandatory	Column Name of the database table to which this property maps to.
ReferenceEntityName	Optional	Name of the entity it refers to. Defaults to the class type of the property.
NotNull	Optional	Specifies whether it can contain null value or not. Defaults to false.
CascadeType	Optional	This specifies how the child records will be affected during CURD operations. Defaults to CascadeType.SaveUpdate. See CascadeType.
Lazy	Optional	Enables or disables lazy loading. Defaults to true. For Example: [ManyToOne("PackageId", NotNull=true)] public virtual Package package { get; set; }

AccessType

It specifies how the data from database is injected into your model.

Property (default) - uses the get/set access of the property.

Field - access the field directly.

NoSetter - access the field for setting data and access the property for getting data.

ClassName – your own implementation.

CascadeType

This specifies how the relations are handled in your model.

All (default)

None

SaveUpdate

Delete

DeleteOrphan

AllDeleteOrphan

In cascade insert/update, the entity model should not contain foreign key column name as its property. Instead it should define the associated entity model.

```
[ManyToOne("Contact_LeadId")]
public virtual LeadDetails LeadDetails { get; set; }
```

In this example, Contact entity has association with Lead details. Hence in contact model the lead details model should be defined as the property instead of foreign key (Contact_LeadId).

CascadeType.All

These settings will insert/update/delete all the associated objects.

```
LeadDetails newleadDetails = new LeadDetails{Name = "new", Contact = new ContactDetails {Email
= "new@new.com" }};
daAutomationService.Add(newleadDetails); // will insert the contact record too.
daAutomationService.Add(newleadDetails); // will update the contact record too.
daAutomationService.Add(newleadDetails); // will delete the contact record too.
```

CascadeType.SaveUpdate

This setting will insert/update all the associated objects when performing insert/update operations.

```
daAutomationService.Add(newleadDetails); //will insert the lead & contact record.
daAutomationService.Add(newleadDetails); //will update the lead & contact record.
daAutomationService.Add(newleadDetails); //will not delete the contact record.
```

CascadeType.Delete

This setting will delete the associated objects while delete operation. If the current entity refers to other entities (foreign key constraint on database), No hard delete will be performed.

```
daAutomationService.Add(newleadDetails); // will insert only lead record.
daAutomationService.Add(newleadDetails); // will update only lead record.
daAutomationService.Add(newleadDetails); // will delete the lead/contact record.
// if contact refers to any other entity it will not delete the contact record
```

CascadeType.None

This setting will not affect the associated objects.

IdentifierDataType

String, Int, Guid

Complete Example

Lead Details

```

/// <summary>
/// This is how business entity has to be created.
/// BaseEntity has to be inherited.
/// We have to specify entity identifier(Which has been entered in DB) in EntityIdentifier attribute.
/// </summary>
[EntityIdentifier(Name = "Lead")]
[Entity("LeadDetails", IdentifierDataType.Guid, "Lead_Id")]
public class LeadDetails : TenantTransactionalEntity
    /// <summary>
    /// Gets or sets the Lead Name
    /// </summary>
    [Simple(ColumnName = "Lead_Name", NotNull = true)]
    public virtual string Name { get; set; }
    /// <summary>
    /// Gets or sets the Lead Status
    /// </summary>
    [Simple(ColumnName = "Lead_LeadStatus", NotNull = true)]
    public virtual LeadStatus LeadStatus { get; set; }
    /// <summary>
    /// Gets or sets the Created On
    /// </summary>
    [Simple(ColumnName = "Lead_CreatedOn", NotNull = true, Update = false)]
    public virtual DateTime CreatedOn { get; set; }

```

```

    /// <summary>
    /// Gets or sets the Created By
    /// </summary>
    [Simple(ColumnName = "Lead_CreatedBy", NotNull = true, Update = false)]
    public virtual string CreatedBy { get; set; }
    /// <summary>
    /// Gets or sets the Updated On
    /// </summary>
    [Simple(ColumnName = "Lead_UpdatedOn", Insert = false)]
    public virtual DateTime UpdatedOn { get; set; }
    /// <summary>
    /// Gets or sets the Updated By
    /// </summary>
    [Simple(ColumnName = "Lead_UpdatedBy", Insert = false)]
    public virtual string UpdatedBy { get; set; }
    /// <summary>
    /// Gets or sets the Status
    /// </summary>
    [Simple(ColumnName = "Lead_Status", NotNull = true)]
    public virtual bool Status { get; set; }
    /// <summary>
    /// Gets or sets the contact Details of Lead
    /// </summary>
    [OneToMany("ContactDetails", "Contact_LeadId", Lazy = false)]
    public virtual ICollection<Contacts> ContactDetails { get; set; }
}
Contacts
    /// <summary>
    /// This is the contacts entity
    /// </summary>

```

```

/// BaseEntity has to be inherited to manage extended fields.
/// We have to specify entity identifier(Which has been entered in DB) in EntityIdentifier attribute.
/// </summary>
[EntityIdentifier(Name = "Contact")]
[Serializable]
[Entity("ContactDetails", IdentifierDataType.Guid, "Contact_Id")]
public class Contacts : TenantTransactionalEntity
{
    /// <summary>
    /// Gets or sets the Name

```

```

    /// </summary>
    [Simple(ColumnName = "Contact_Name")]
    public virtual string Name { get; set; }
    /// <summary>
    /// Gets or sets the Designation
    /// </summary>
    [Simple(ColumnName = "Contact_Designation")]
    public virtual string Designation { get; set; }
    /// <summary>
    /// Gets or sets the Gender
    /// </summary>
    [Simple(ColumnName = "Contact_Gender")]
    public virtual string Gender { get; set; }
    /// <summary>
    /// Gets or sets the Phone Number
    /// </summary>
    [Simple(ColumnName = "Contact_Phone")]
    public virtual string PhoneNumber { get; set; }
    /// <summary>
    /// Gets or sets the Email ID
    /// </summary>
    [Simple(ColumnName = "Contact_EmailId")]
    public virtual string Email { get; set; }
    /// <summary>
    /// Gets or sets the created by.
    /// </summary>
    [Simple(ColumnName = "Contact_CreatedBy", NotNull = true, Update = false)]
    public virtual string CreatedBy { get; set; }
    /// <summary>
    /// Gets or sets the created on.
    /// </summary>
    [Simple(ColumnName = "Contact_CreatedOn", NotNull = true, Update = false)]
    public virtual DateTime CreatedOn { get; set; }
    /// <summary>
    /// Gets or sets the updated by.
    /// </summary>
    [Simple(ColumnName = "Contact_UpdatedBy", Insert = false)]
    public virtual string UpdatedBy { get; set; }
    /// <summary>
    /// Gets or sets the updated on.
    /// </summary>
    [Simple(ColumnName = "Contact_UpdatedOn", Insert = false)]
    public virtual DateTime UpdatedOn { get; set; }
    /// <summary>
    /// Gets or sets the status.
    /// </summary>
    [Simple(ColumnName = "Contact_Status")]

```

```

public virtual bool Status { get; set; }
/// <summary>
/// Gets or sets Leade Details
/// </summary>
[ManyToOne("Contact_LeadId")]
public virtual LeadDetails LeadDetails { get; set; }
}

```

DataAccessAutomationService API

The IDataAccessAutomationService interface exposes following API's:

`string Create(TransactionalBaseEntity entity)`

Populate the entity object with necessary values and call this service, it will automatically add the record to your configured database. It will insert child relations records too if Cascade property (All/SaveUpdate) is specified.

`void Update(TransactionalBaseEntity entity)`

Update the entity object and call this service, it will update the corresponding record in the database. It will update the child relations too if Cascade property (All/SaveUpdate) is specified.

`T Fetch(object identifier)`

Call this service with the record/entity identifier (int/string/guid). It will automatically fetch the matching record from the database and populate the corresponding entity then returns it. This will fetch the relational objects in the entity.

`T ShallowFetch(object identifier)`

Call this service with the record/entity identifier (int/string/guid). It will automatically fetch the matching record from the database and populate the corresponding entity then returns it. This will not fetch the relational objects in the entity.

`void Delete(object identifier)`

Call this service with the entity identifier if you want to delete the entity in the database. It will permanently delete the record. It will delete the one-to-one & many-to-one relations if the cascade property (All/Delete/DeleteOrphan) is specified.

`Dictionary<string, T>ShallowSearch(string condition)`

This service will search for the base entity with the given search condition. Here child associated entity details will not be available.

CelloSaaS will contain the Fill Child Entities method in the final version.

`Dictionary<string, T>ShallowSearch(SearchParameters)`

This service will search for the limited number of base entity with the given search condition. Search Parameter contains the start record number, end record number, sort string and sort direction details. Here child associated entity details will not be available.

`Dictionary<string, T> FillChildDetails <T1> (Dictionary<string, T> parentDetails, string referenceColumnName, string childPropertyName) where T1 : TransactionalBaseEntity`

This service is used to fill child details of parent details. This service is used to fill collection type child details.



Don't pass more than 100 parent details because this could affect the performance of the application.

```

IDataAccessAutomationService<T> CelloSaaSDALAutomationService =
(IdataAccessAutomationService<T>)ServiceLocator.GetServiceImplementation(typeof(IDataAccess
AutomationService<T>));
LeadDetails leadDetails = new LeadDetails{
Name = "Test", LeadStatus = LeadStatus.New, CreatedBy = "Admin",
}

```

```
CreatedOn = DateTime.Now, Status = true      };
    // save the lead details to the database
    string leadIdentifier = CelloSaaSDALAutomationService.Create(leadDetails);
    // fetch the lead from database
    var lead = CelloSaaSDALAutomationService.Fetch(leadIdentifier);
    lead.Name = "Test_1";
    lead.LeadStatus = LeadStatus.Success;
    // update the changes to the database
    CelloSaaSDALAutomationService.Update(lead);
    // delete the lead from the database
    CelloSaaSDALAutomationService.Delete(leadIdentifier);
```

5 SERVICE FACTORIES AND DAL IMPLEMENTATION FACTORIES

CelloSaaS follows interface driven programming and advocates loose coupling. For this purpose, all the instances of services and DAL should be obtained via ServiceFactories and DALImplementationFactories respectively. The factories in turn uses service dependency injection container Unity. Service factories are used to create instance for service. CelloSaaS.SaaS.Library.ServiceLocator's "GetService Implementation" method returns the service instance based on the service interface type.

```
CelloSaaSSaaS.Library.ServiceLocator
///<summary>
///This method is used to return service instance for passing service interface type
///</summary>
///<param name="interfaceType">Service Interface Type</param>
///<returns>Service instance</returns>
public static object GetServiceImplementation(Type interfaceType)
```

Example: create UserDetailsServiceinstance

```
CelloSaaSSaaS.ServiceContracts.UserManagement.IUserDetailsService
userDetailsService=(CelloSaaSSaaS.ServiceContracts.UserManagement.IUserDetailsService)
CelloSaaSSaaS.Library.ServiceLocator.GetServiceImplementation(typeof(CelloSaaSSaaS.ServiceContracts.UserManagement.IUserDetailsService));
```

DAL Implementation factories are used to create instance for DAL.CelloSaaSSaaS.Library.DataAccessLayer.DALImplementationFactory's "Get DALImplementation" method will return the DAL instance based on the DAL interface type

```
CelloSaaSSaaS.Library.DataAccessLayer.DALImplementationFactory
///<summary>
/// Returns the respective DAL implementation based on the provider
/// The respective provider implementation should follow a naming convention
///</summary>
///<param name="t">DAL Interface Type</param>
///<returns>DAL instance</returns>
public static object GetDALImplementation(Type t)
```

```
To create UserDetailDALinstance
CelloSaaS.DAL.UserManagement.IUserDetailDAL
userDetailsDAL=(CelloSaaS.DAL.UserManagement.IUserDetailDAL)
CelloSaaSSaaS.Library.DataAccessLayer.DALImplementationFactory.GetDALImplementation(typeof(CelloSaaS.DAL.UserManagement.IUserDetailDAL));
```

In web.config, add a new section to allow for Unity to be configured.

```
<sectionname="unity" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection,
Microsoft.Practices.Unity.Configuration" />
Registering the new section will allow you to set the path to the Unity configuration file.
<unityconfigSource="Unity.config" />
```

In unity config, following sections are important to register the interface.

Example

```
<typeAlias alias="ITenantDAL"
           type="CelloSaaS.DAL.TenantManagement.ITenantDAL, CelloSaaS.DAL" />
```

Alias name should be an interface name and type is interface type name. Register the class that implements the interface.

Example

```
<typeAlias alias="TenantDAL"
           type="CelloSaaS.SqlDAL.TenantManagement.TenantDAL, CelloSaaS.SqlDAL" />
```

Set name of the class as the Alias name and assembly details as type, then map the interface to the implemented class.

Example

```
<type type="ITenantDAL" mapTo="TenantDAL" name="ITenantDAL">
```

The 'type' and name represent the Interface name and 'mapTo' represents implementation class name.

Example

```
xmlversion="1.0"encoding="utf-8" ?>
<unity>
<typeAliases>
<typeAliasalias="ITenantDAL"
type="CelloSaaS.DAL.TenantManagement.ITenantDAL, CelloSaaS.DAL" />
<typeAliasalias="TenantDAL"
type="CelloSaaS.SqlDAL.TenantManagement.TenantDAL, CelloSaaS.SqlDAL" />
<typeAliasalias="ITenantService"
type="CelloSaaS.ServiceContracts.TenantManagement.ITenantService,
CelloSaaS.ServiceContracts" /><typeAliasalias="TenantService"
type="CelloSaaS.Services.TenantManagement.TenantService, CelloSaaS.Services" />
</typeAliases>
<containers>
<containername="DataAccess">
<types>
<typetype="ITenantDAL"mapTo="TenantDAL"name="ITenantDAL">
<typeConfigextensionType="Microsoft.Practices.Unity.Configuration.TypeInjectionElement,Microsoft
.Practices.Unity.Configuration">
</typeConfig>
</type>
<typetype="ITenantService"mapTo="TenantService"name="ITenantService">
<typeConfigextensionType="Microsoft.Practices.Unity.Configuration.TypeInjectionElement,Microsoft
.Practices.Unity.Configuration">
</typeConfig>
</type>
</types>
</container>
</containers>
</unity>
```

5.1 Caching

Caching is a simple concept that takes dynamic content and then creates a temporary reserve of that information. That temporary content is then served as static content until it expires. The types of caching mechanisms differentiate on what level they cache the data.

CelloSaaS leverages caching extensively for the following components. They are

DataScope

EntityBasedDataScope

Privilege

EntityMetaData

ExtendedFields

BaseFields

TenantLicense

Package

TenantSettings

TenantRelation

DataView

GlobalDefaultValues

GlobalFields

DataViewFieldDefaultValues

DataViewMetaData

TenantSettings

Roles

MetaData

Tenant Hierarchy

5.1.1 AppFabric Cache

AppFabric Caching is a distributed caching mechanism that allows multiple web applications to use the same cache, this is different to how most websites currently work, and even Orchard by default maintains its own cache within its separate website instances. With AppFabric, all website instances maintain and contribute to the same cache.

By using cache, application performance can improve significantly by avoiding unnecessary calls to the data source. Distributed cache enables your application to match increasing demand with increasing throughput using a cache cluster that automatically manages the complexities of load balancing. When you use cache you can retrieve data by using keys or other identifiers, named "tags." App Fabric Cache supports optimistic and pessimistic concurrency models, high availability, and a variety of cache configurations. "App Fabric Cache" includes an ASP.NET session provider object that enables you to store ASP.NET session objects in the distributed cache without having to write to databases, which increases the performance and scalability of ASP.NET applications.

☞ Download AppFabric from the following link based on caching deployment machine.

Platform

Setup Package

Windows Vista and Windows Server 2008 x64	WindowsServerAppFabricSetup_x64_6.0.exe
Windows 7 and Windows Server 2008 R2 x64	WindowsServerAppFabricSetup_x64_6.0.exe
Windows Vista and Windows Server 2008 x86	WindowsServerAppFabricSetup_x86_6.0.exe
Windows 7 x86	WindowsServerAppFabricSetup_x86_6.1.exe

☞ Download Link :<http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=15848>

Installation Steps:

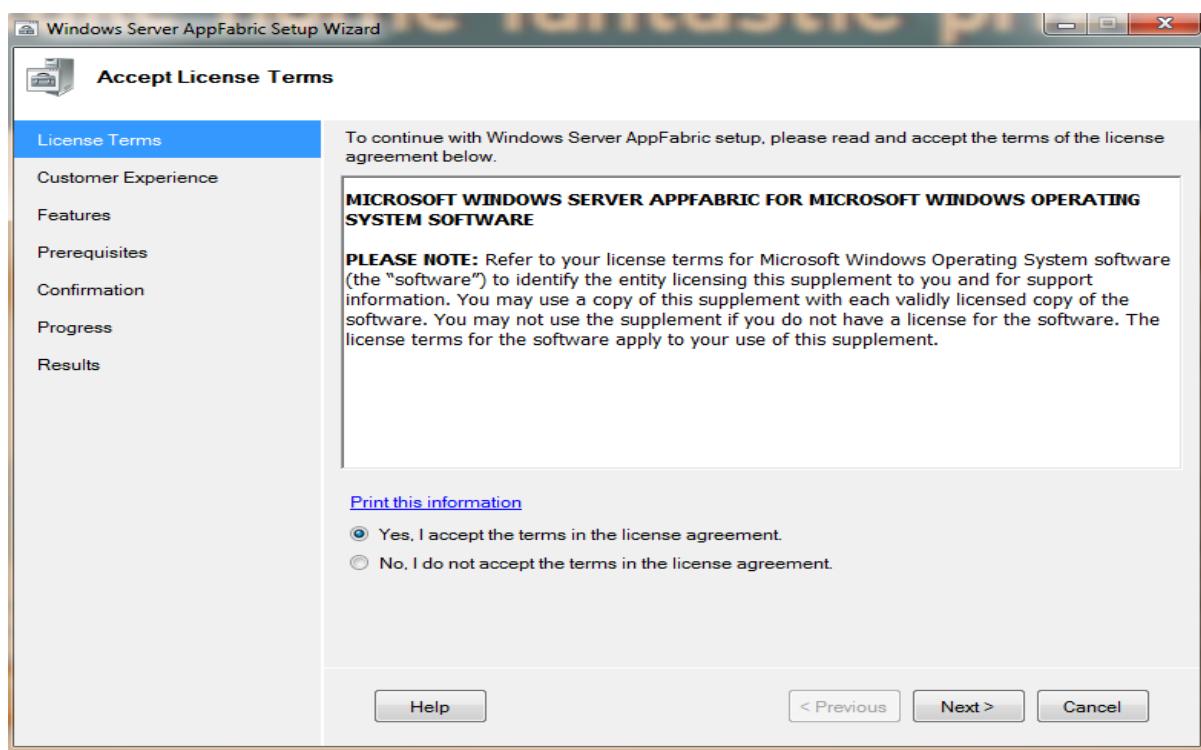


Figure 5-1 - Accept License Terms Screen

☞ Accept the licence agreement and click “Next>”.

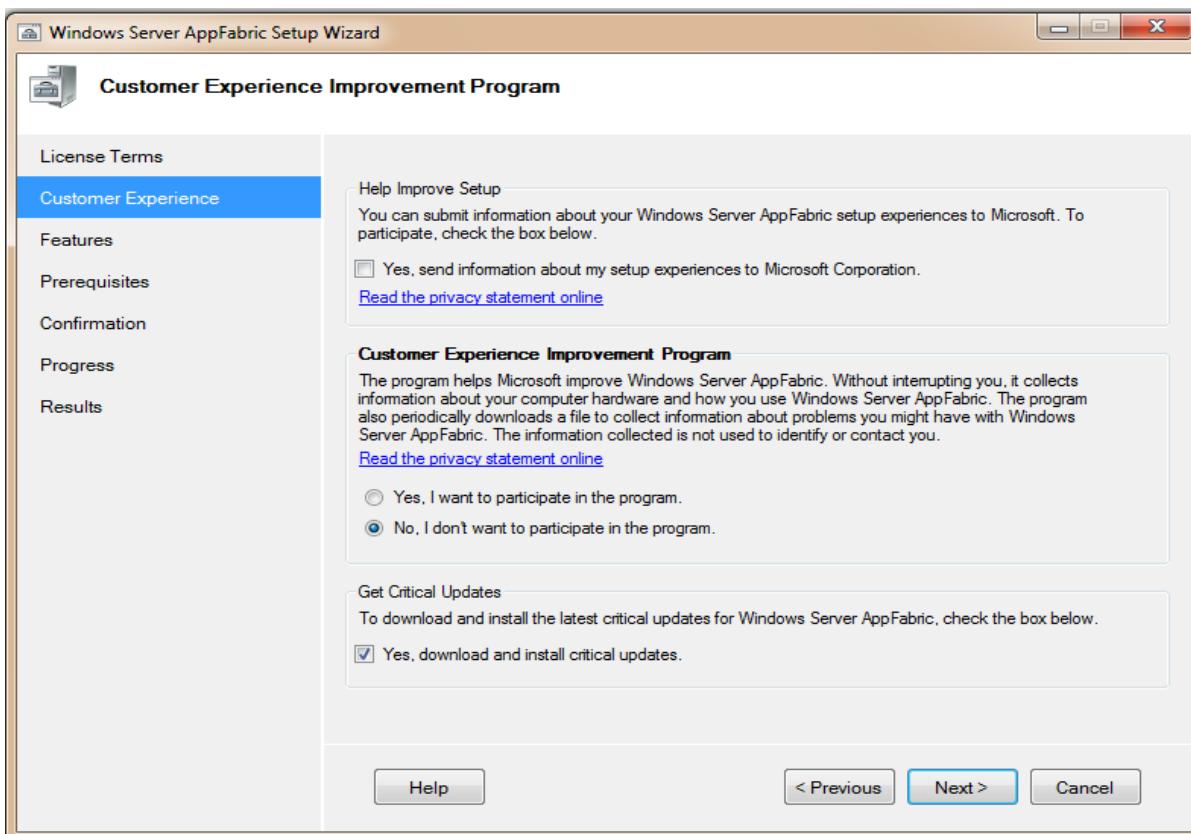


Figure 5-2– Customer Experience Screen

☞ Click “Next>”.

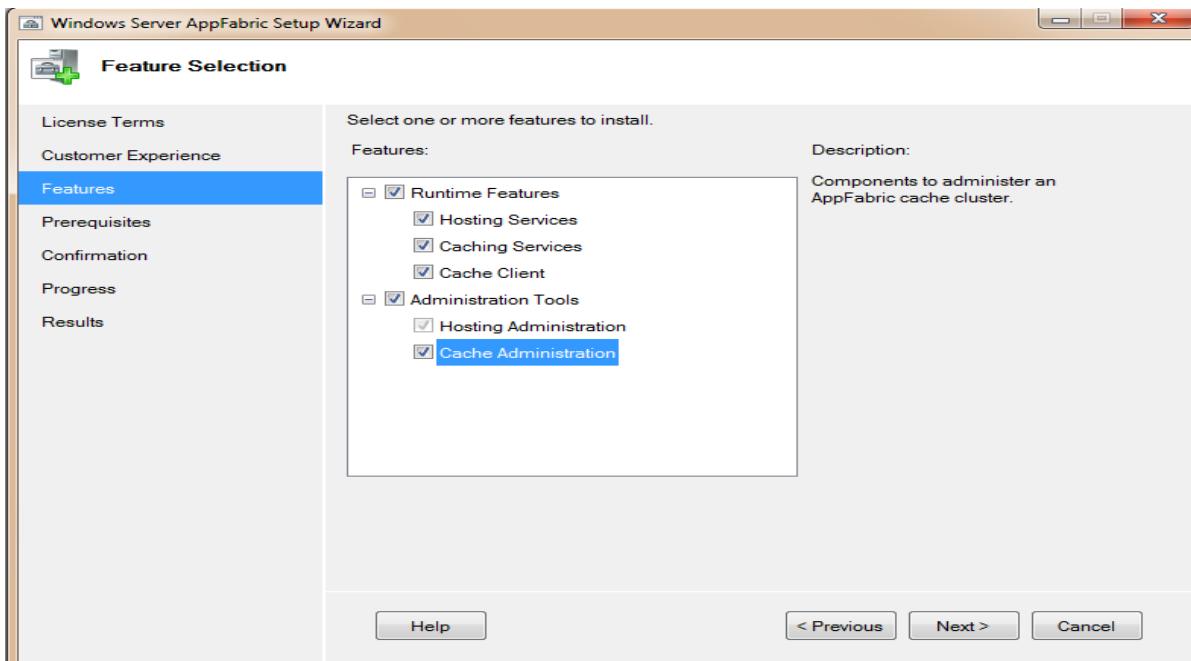


Figure 5-3– Features Screen

☞ Select all options and click “Next>”.

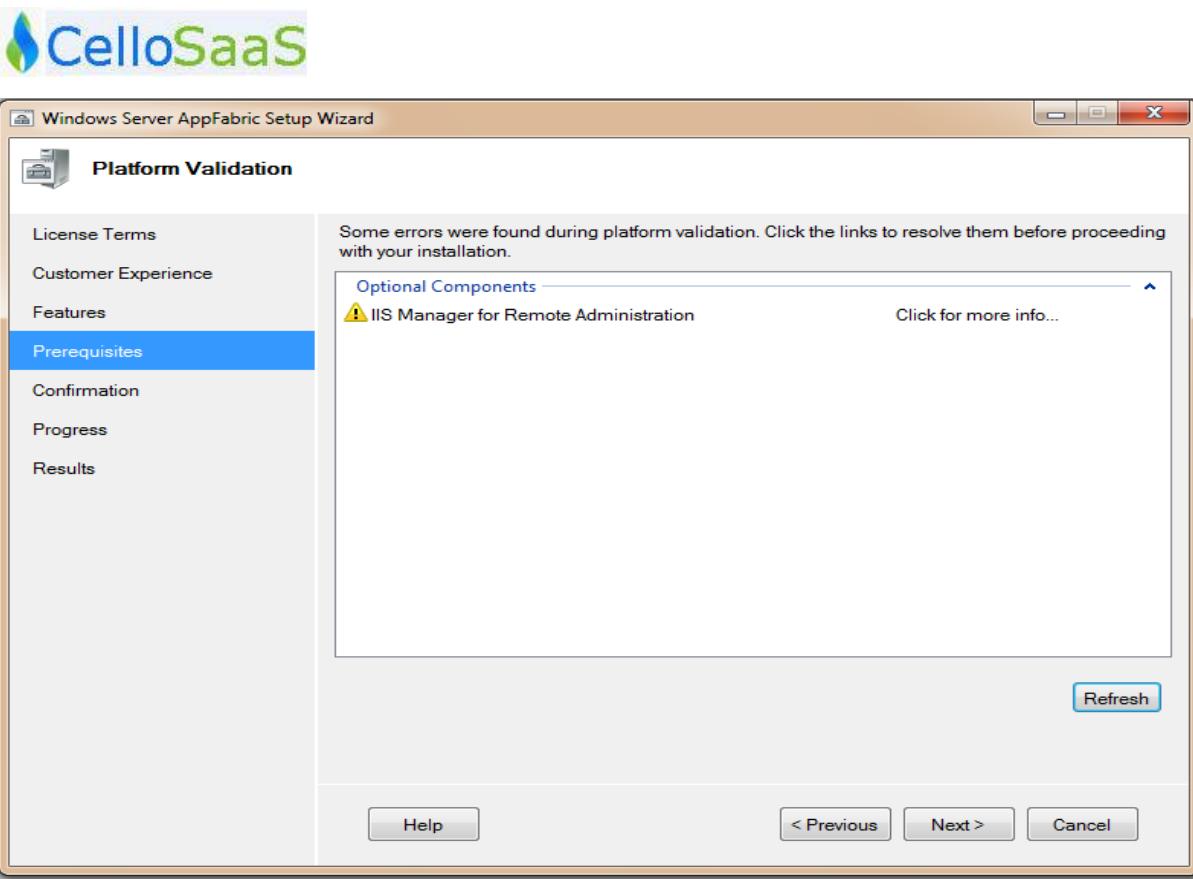


Figure 5-4– Prerequisites Screen

☞ Click “Next>”.

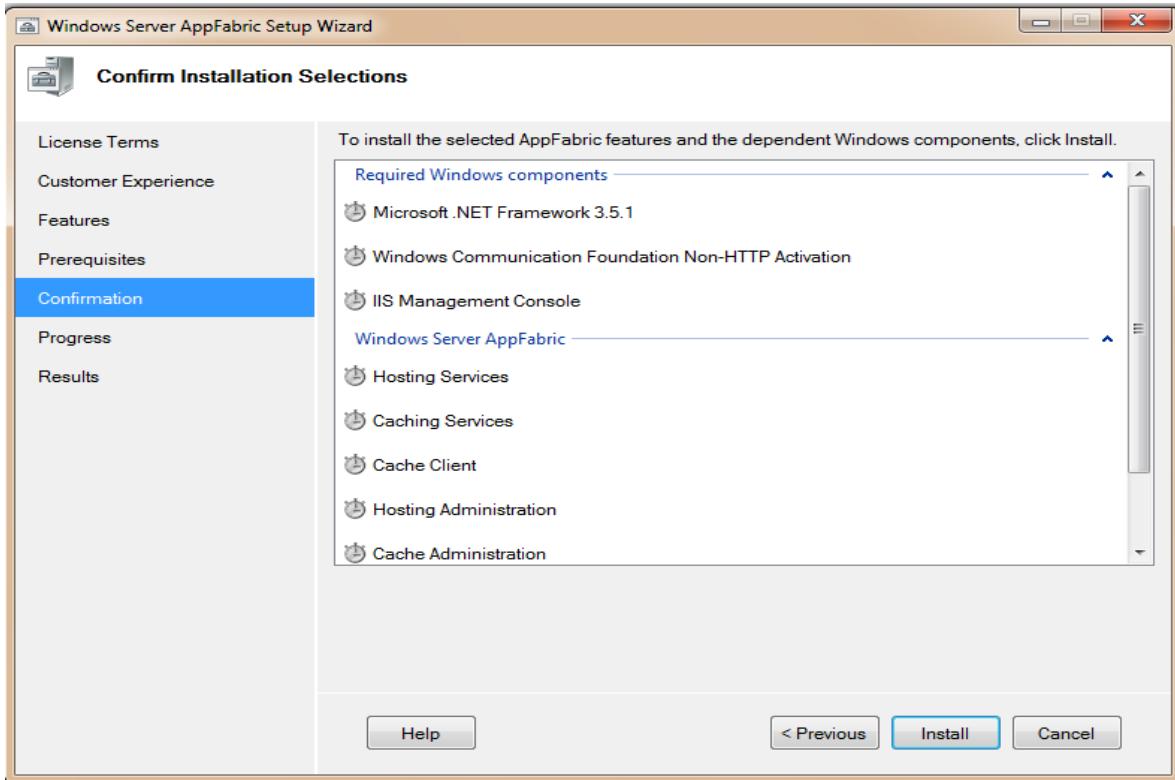


Figure 5-5– Confirmation Screen



☞ Click “Install”.

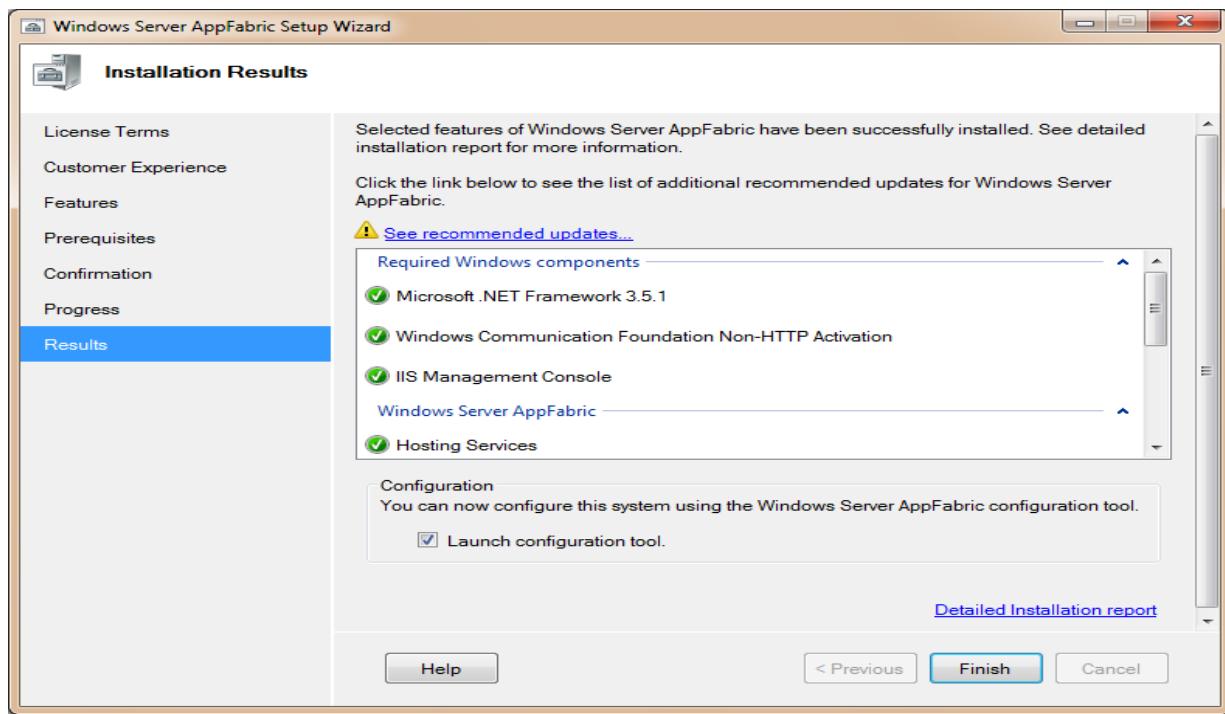


Figure 5-6– Results Screen

☞ Click “Finish”. Then AppFabric Configuration Wizard will be available to configure cluster.

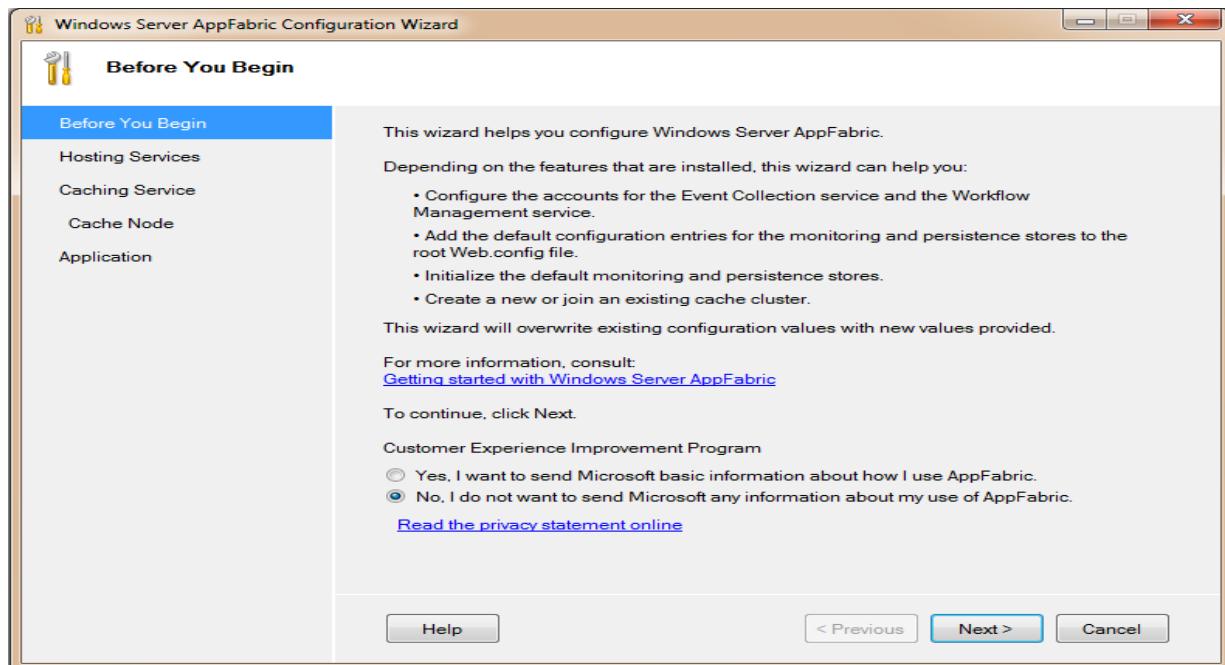


Figure 5-7 – Before You Begin Screen



☞ Click “Next>”.

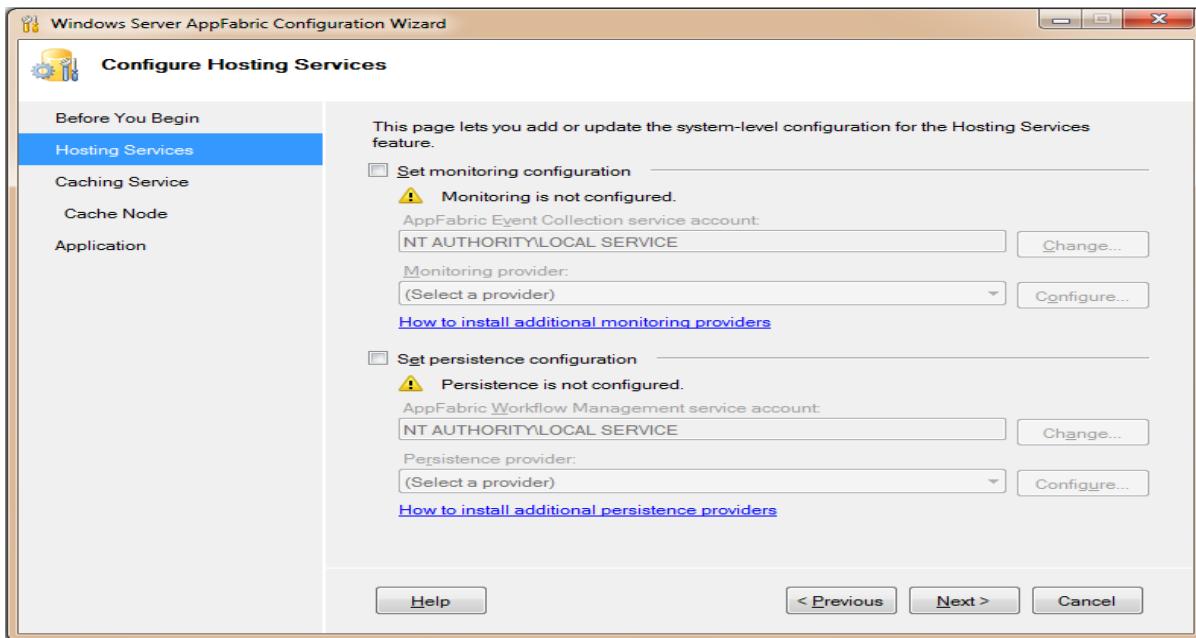


Figure 5-8– Hosting Services Screen

☞ Click “Next>”.

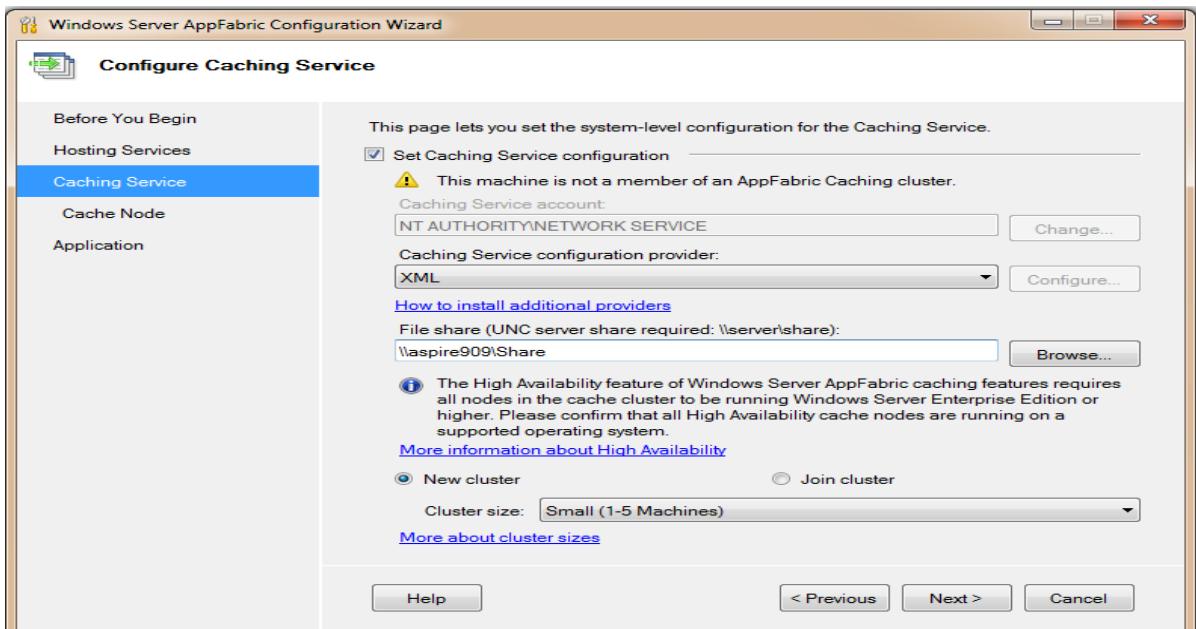


Figure 5-9– Caching Services

☞ Check “Select Caching Service Configuration” and select “XML” in Caching Service Configuration provider”.

- ☞ Give the valid shared folder name in “File Share” path.
- ☞ Select the required Cluster Size from the dropdown (Small, Medium and Large).

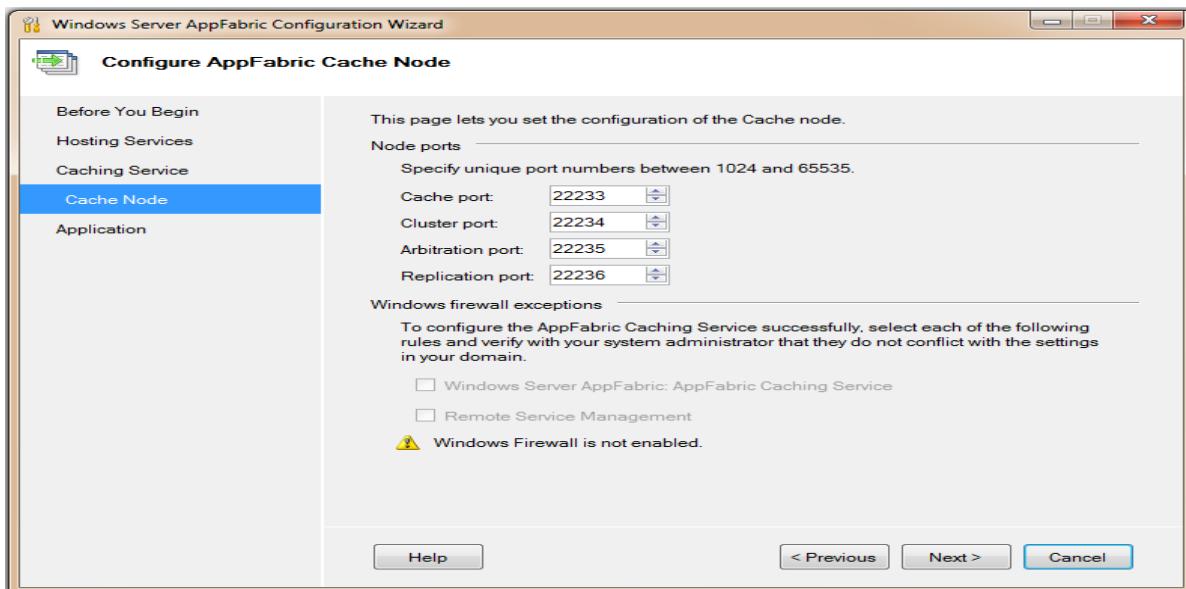


Figure 5-10– Cache Node Screen

- ☞ Click “Next>”.

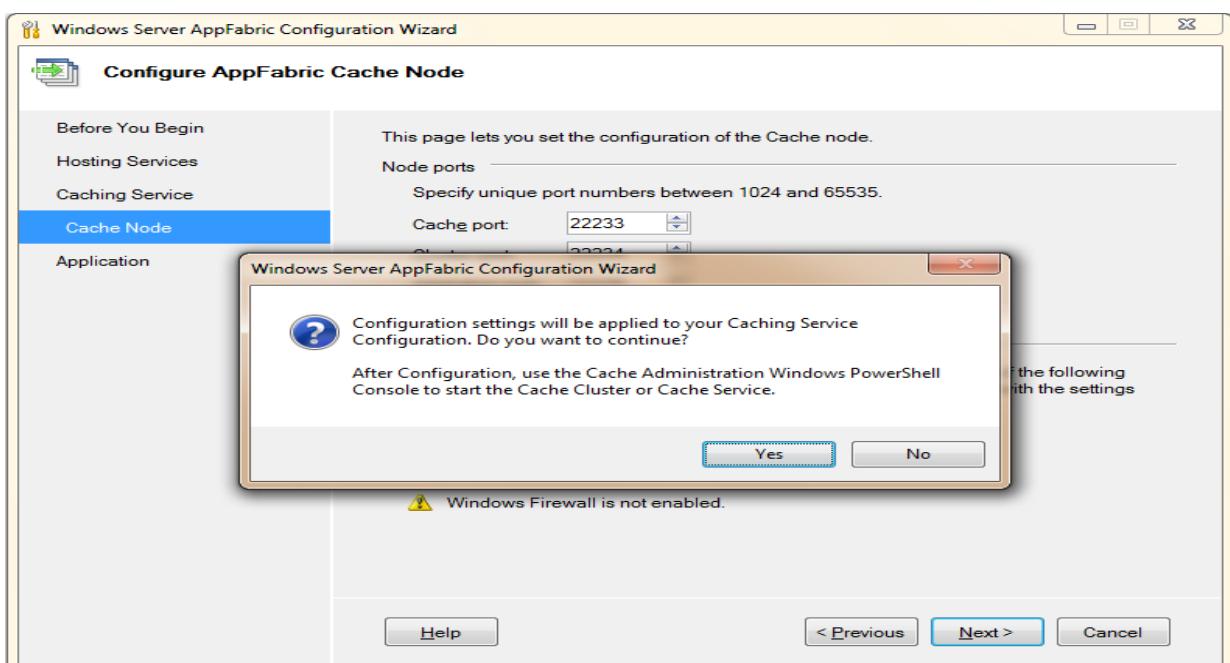


Figure 5-11– Windows Server AppFabric Configuration Wizard Screen

- ☞ Select “Yes”.

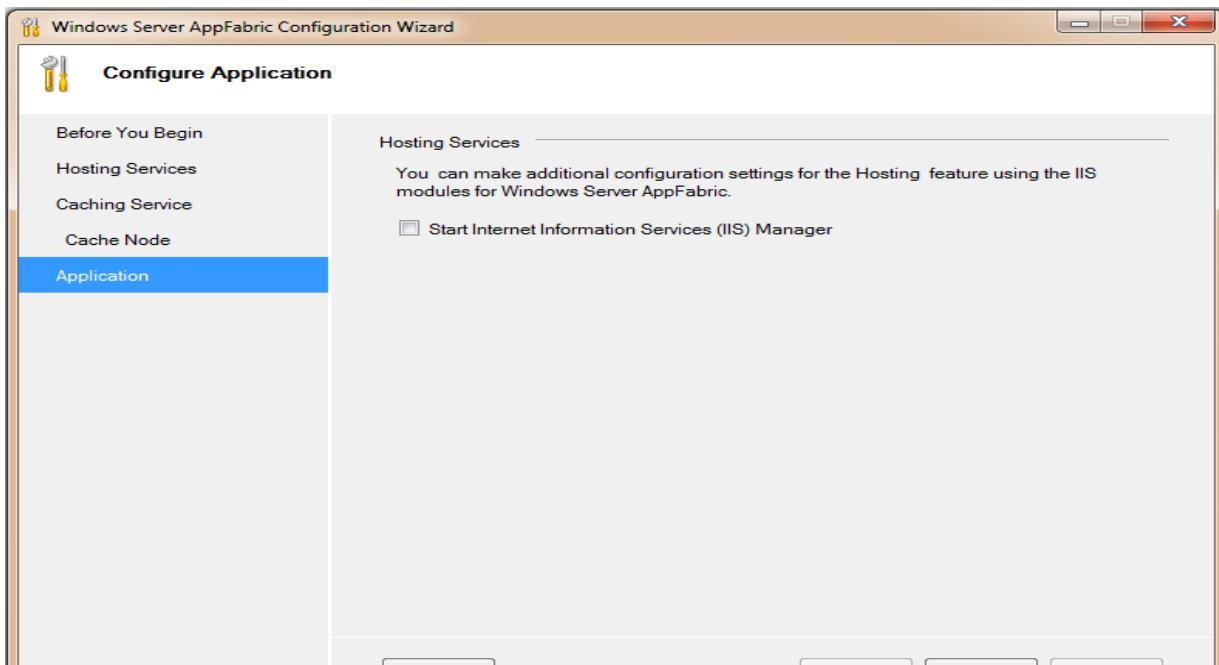


Figure 5-12– Configure Application Screen

- ☞ Click “Finish”.
- ☞ Run Caching Administration Windows PowerShell. And Start the Cache cluster using the command “Start-CacheCluster”.

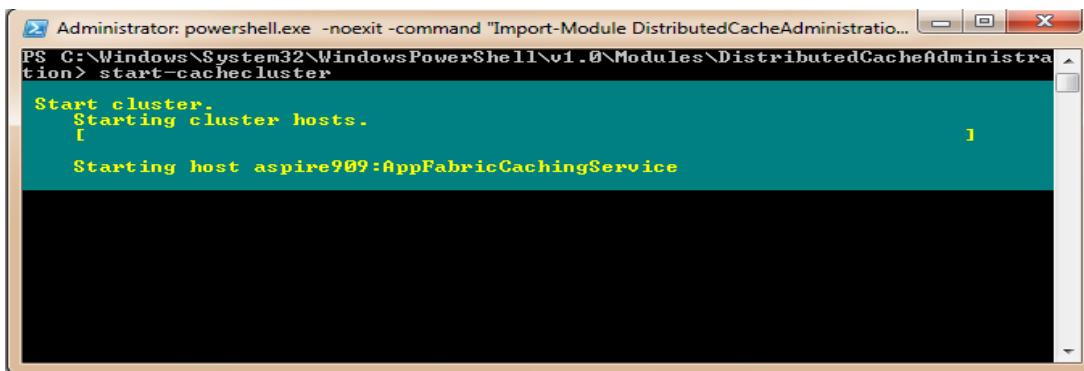


Figure 5-13– Cache Cluster Screen

5.1.2 How to use in Application

- ☞ Add the following Dll files to the CelloSaaSDll folder if these does not exist already.
 - a. Microsoft.WindowsFabric.Common.dll
 - b. Microsoft.WindowsFabric.Data.Common.dll
 - c. Microsoft.ApplicationServer.Caching.Client.dll

d. Microsoft.ApplicationServer.Caching.Core.dll

- ☞ And add references to these four dll files in the WebApplication
- ☞ In the web.config file, add the following configuration settings for enabling the AppFabric Caching

```
<cachingConfiguration defaultCacheManager="AppFabric Cache Manager">
    <cacheManagers>
        <add name="AppFabric Cache Manager"
            type="CelloSaaS.Library.Provider.AppFabricCacheManager, CelloSaaS.Library"
            RoutingClient="false" LocalCache="false" HostName="hostname"
            CachePort="22233" CacheHostName="DistributedCacheService" NamedCache="default"
            InvalidationPolicy="TimeoutBased" DefaultTimeOut="10" SecurityMode="None"
            ProtectionLevel="None"/>
    </cacheManagers>
</cachingConfiguration>
```

- ☞ Comment out the other cachingConfiguration in the web.config file
- ☞ Replace the HostName attribute value with the server name in which appfabric caching is enabled
- ☞ Appfabric Property details.

Property	Mandatory	Description	Default Value
RequestTimeout (seconds)	NO	Used to set the Client time-out. Microsoft does not recommend specifying a value less than 10000 (10 seconds).	3(15)
ChannelOpenTimeout (seconds)	NO	This attribute value can be set to 0 in order to immediately handle any network problems.	15(3)
MaxConnectionsToServer	NO	Used to set the Maximum number of connections to the server.	1
HostName	Yes	Specify the Host Name	
CachePort	Yes	Specify the Cache Port	
CacheHostName	Yes	Specify the Cache Host Name	
NamedCache	Yes	Specify the Cache Name	
LocalCache	Yes	Used to Enable the local cache. Possible values "True" and "False".	
InvalidationPolicy	Yes	Possible values include NotificationBased and TimeoutBased.	
DefaultTimeOut (seconds)	Yes	Local cache time-out.	
ObjectCount	NO	Maximum locally-cached object count	10,000
PollInterval	NO	Specific cache notifications poll interval (seconds)	300
MaxQueueLength	NO	Maximum queue length	10,000
SecurityMode	NO	Possible values include Transport and None.	Transport

ProtectionLevel	NO	Possible values include None, Sign, and EncryptAndSign.	EncryptAndSign
ChannelInitializationTimeout (seconds)	NO	Channel initialization timeout	
ConnectionBufferSize	NO	Connection buffer size	
MaxBufferPoolSize	NO	Maximum buffer pool	
MaxBufferSize	NO	Maximum buffer size	
MaxOutputDelay(seconds)	NO	Maximum output delay	
ReceiveTimeout(seconds)	NO	Receive timeout	

AddToCache method

```
Dictionary<string,string> messages= new Dictionary<string,string>();
    messages.Add("suggestion!@$!e_suggestion_load", "Suggestions could not be loaded.");
    messages.Add("suggestion!@$!e_suggestion_insert", "Suggestions has not been
inserted.");
    messages.Add("suggestion!@$!e_suggestion_insert_Success", "Suggestions has been
successfully inserted.");
    CacheManager.AddToCache("messageCacheKey", messages);
```

RemoveFromCache method

```
if (CacheManager.GetData("messageCacheKey") != null)
    CacheManager.RemoveFromCache("messageCacheKey");
```

5.1.3 Advantages of Caching

The two main importance of caching are:

To reduce latency: Because the request is satisfied from the cache (which is closer to the client) instead of the origin server, it takes less time for it to get the representation and display it. This makes the Web seem more responsive.

To reduce network traffic: Because representations are reused, it reduces the amount of bandwidth used by a client. This saves money if the client is paying for traffic, and keeps their bandwidth requirements lower and more manageable.

5.1.4 Caching Best Practices

The following are some of the best practices in caching:

Store the cached data in a variable:

- This is a design pattern that ensures that the cached object will not be destroyed (removed) while you are working with it. By setting a reference to the object first (a variable), you ensure that there is no change or disappearance at some random point in your code.

When caching large amount of data, enable the cleanup (flush) mechanisms:

Cache includes cleanup mechanisms for these reasons:

- To make it easy for the developer to specify the lifetime of cached data
- To limit the memory and resources consumed by cached data
- To ensure that the most-used items remain in the cache, while the least-used items are evicted from the cache.

Cache data in the most useful format:

- The goal here is to cache your data in a format that is immediately useful when a page

Members of Caching Class

- Before moving in depth to caching class members, we will take a look at how caching is configured in CelloSaaS. The configuration of caching is added in cello config file under <cachingConfiguration> tag as follows:

```
<cachingConfiguration defaultCacheManager="Velocity Cache Manager">
    <cacheManagers>
        <add name="Velocity Cache Manager"
            type="CelloSaaS.Library.Provider.VelocityCacheManager, CelloSaaS.Library"
            RoutingClient="false" LocalCache="true" HostName="192.168.2.106"
            CachePort="22233" CacheHostName="DistributedCacheService"
            NamedCache="default" />
    </cacheManagers>
</cachingConfiguration>
<cachingConfiguration defaultCacheManager="Cache Manager">
    <cacheManagers>
        <add expirationPollFrequencyInSeconds="1440"
            maximumElementsInCacheBeforeScavenging="1000"
            numberToRemoveWhenScavenging="10" backingStoreName="Null Storage"
            type="Microsoft.Practices.EnterpriseLibrary.Caching.CacheManager,
            Microsoft.Practices.EnterpriseLibrary.Caching, Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=31bf3856ad364e35"
            name="Cache Manager" />
    </cacheManagers>
    <backingStores>
        <add encryptionProviderName=""
            type="Microsoft.Practices.EnterpriseLibrary.Caching.BackingStoreImplementations.NullBackingStor
            e, Microsoft.Practices.EnterpriseLibrary.Caching, Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=31bf3856ad364e35"
            name="Null Storage" />
    </backingStores>
</cachingConfiguration>
```

The two considerations when it comes to caching implementation are:

How to achieve caching

When to remove the content from cache

The below methods address these two consideration and helps implementing cache in CelloSaaS.

Namespace - CelloSaaS.Library

Class - CacheManager

AddToCache

Caches store items that are either expensive to create or expensive to transport. Call AddToCache method and pass in the Key, Value to store the data.

public static void AddToCache(string key, object value)

Parameters

Key – specifies the cache key value.

Obj – specifies the cache object to be added.

Above method will add the data to cache and by default set its priority as normal. During removal of cache item (in order to reduce memory consumption), cache will remove items of low priority.

Priority needs to be high for few data due to its frequent access. Hence in CelloSaaS, data with



high level of importance will be added to the cache by setting the following additional parameters to the AddToCache overload method

Item priority

Event action call when object in cache expires
When to expire the item from the cache

Overload Method

```
public static void AddToCache(string key, object value, CacheItemPriority itemPriority,  
ICacheItemRefreshAction cacheItemRefreshAction, params ICacheItemExpiration[]  
cacheItemExpiration)  
{  
}
```

Parameters:

Key - specifies the cache key value.

Value – specifies the cache object to be added.

ItemPriority – specifies the priority of the cache item.

CacheItemRefreshAction – specifies the event action that need to be called when the object in cache expires.

CacheItemExpiration – specifies expirations which dictates when the object should be expired from cache.

CacheManager.RemoveFromCache Method:

Once the lifetime of the data gets expired, cello will remove the data from the cache using RemoveFromCache method.

Namespace: CelloSaaS.Library

Class: CacheManager

Method: public static void RemoveFromCache(string key)

Parameter:

key - specifies the cache key value that need to be removed from the cache

6 TENANT CONFIGURATION FRAMEWORK

In a multi tenant environment, typically single instance serves all the tenants, and hence it is mandatory for the product to provide multi level customization to support the individual tenant's needs. For example, in a typical HR performance appraisal system, company A may want to have 5 point rating and company B may want to have 10 point rating. You should be able to accommodate such needs through configuration. CelloSaaS provides a complete customization framework which allows the developers to build customization into their product easily.

Tenant access to the application

CelloSaaS supports custom urls for each Tenant to access the application. Based on the URL, CelloSaaS takes care of identifying the tenants and their respective theme settings and logos automatically. If there are no settings available for tenant, the default setting will be applied in the application.

Product admin will create URL at the time of Tenant Creation. It should be a valid URL:

Wild Card URL mapping

The URL like `http://*.ApplicationName.com`

The * will be replaced by Tenant Name or some other name

But the application name is same

Example: `http://emc.silica.com`, <http://dazzle.silica.com> (silica.com is the SaaS product name)

Alias URL mapping

Product admin create URL for tenant and then that URL will be mapped to the application.

It should be valid available URL

Example: `http://silica.company.com`, <http://saas.dazzle.com>

The URL needs to be registered by the saas provider and will take care of the dnc entry.

Usage

URL of the tenant can be set when a tenant is created using the tenant management screen. The following method is used to get tenant details by URL.

```
///<summary>
/// This method is to get the tenant details by passing URL
///</summary>
///<param name="url">Tenant Url</param>
///<returns>Tenant Details</returns>
public TenantDetails GetTenantDetailsByURL(string url)
```

```
<% CelloSaaS.Model.TenantManagement.TenantDetails tenantDetails =
CelloSaaS.ServiceProxies.TenantManagement.TenantProxy.GetTenantDetailsByURL(Request.Url.
ToString());
```

```
Following method needs to be used to access the tenant's theme
if (tenantDetails != null && !string.IsNullOrEmpty(tenantDetails.TenantCode))
    Response.Write(Html.Theme(tenantDetails.TenantCode));
else
    Response.Write(Html.Theme(null));
```

```
%>
To access the tenant's logo
Html.Logo(string tenantCode).
<%
if (tenantDetails != null && !string.IsNullOrEmpty(tenantDetails.TenantCode))
    Response.Write(Html.Logo(tenantDetails.TenantCode));
else
```

```
Response.Write(Html.Logo(null));  
%>
```

The above method will return the html image tag. If a tenant logo is available, it will return tenant logo otherwise will return default logo.

Example

In the following example, default URL is <http://silica.company.com>. When you access the application through default URL, you will get the default logo and default themes.

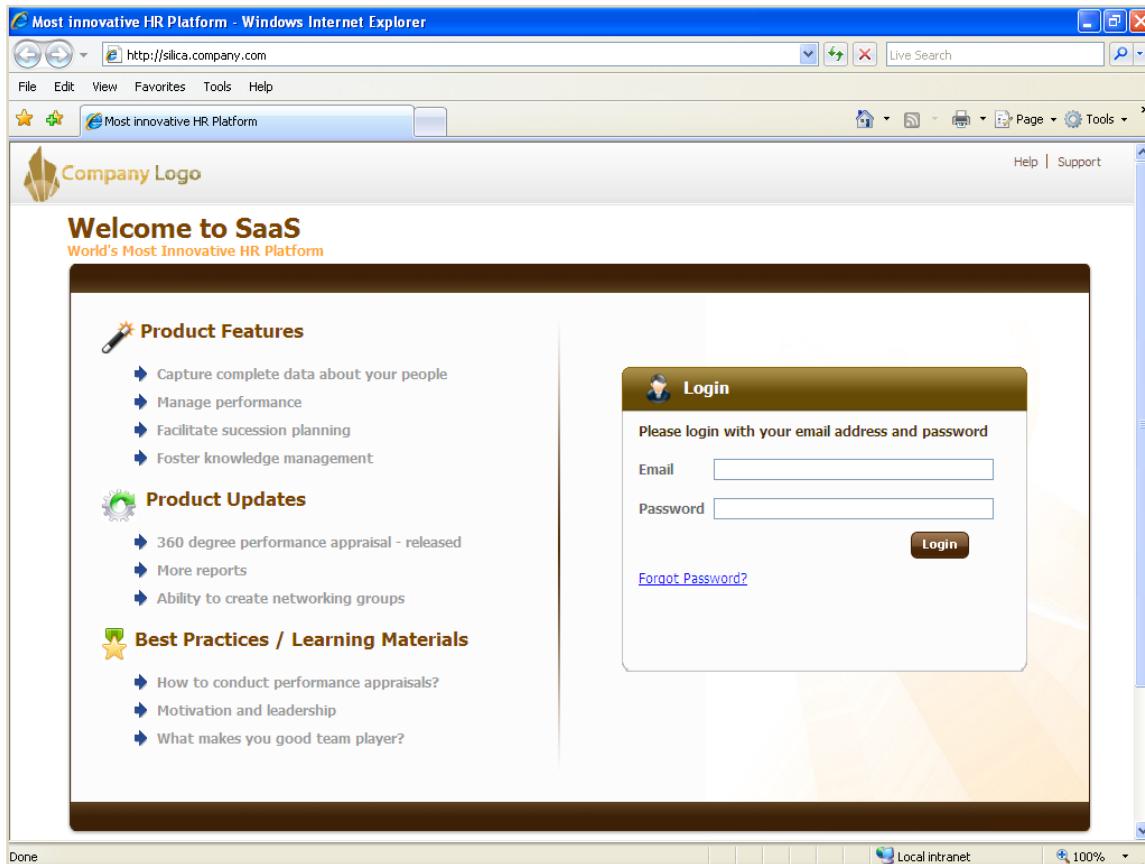


Figure 6-1 Login Page

If you are accessing through Dazzle Tenant URL, then Dazzle logo and their theme will be applied in the application.

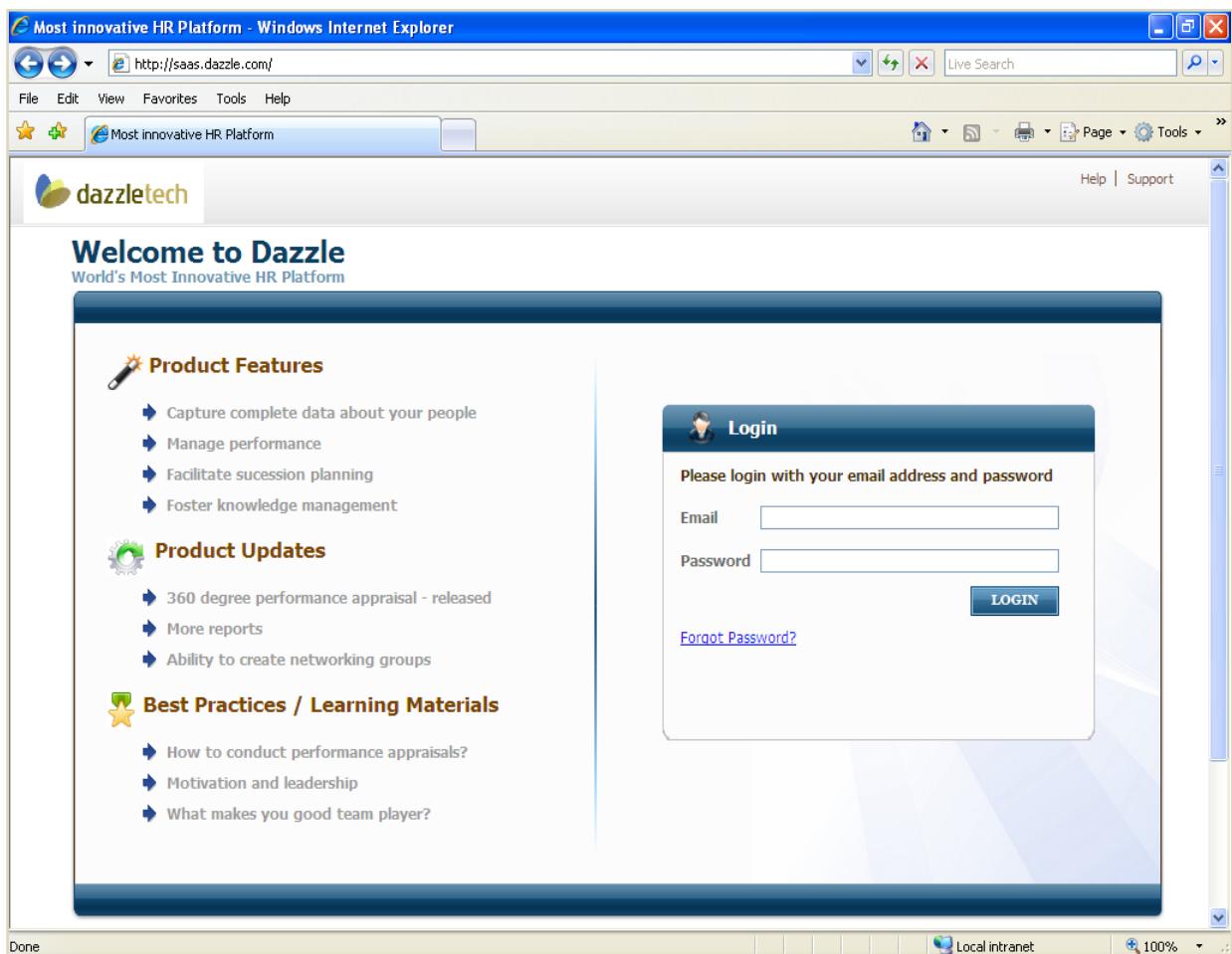


Figure 6-2– Login Page

6.1 Data Model Extension

Every Tenant has unique needs to be addressed with respect to the data they capture and process. While the product can be designed to capture the core or basic data, it needs to offer the ability to extend this data according to the tenant's requirement.

For example, in a HR system, you may want to extend the employee information based on a tenant's requirements.

In company A, they might want to capture the passport details of each employee.

In company B, they might want to capture medical insurance details of each employee.

In these cases, these additional details would be decided by a tenant during the implementation and SaaS application should have the ability to extend the data model defined for a customer. In order to support this, CelloSaaS offers the feature "Data Model Extension" which enables the developers to bring in the data extension support very easily.

Broadly there are different ways of implementing the Data Model Extention. They are

Key Pair Value Table

Flat Table

6.1.1 Key Pair Value Implementation

Key value pair's model can be used to store the extended fields and the data and thus is not limited in terms of the level of data extension. To extend your business data model by using CelloSaaS, you will have to follow the pattern recommended by CelloSaaS.

Model

You will have to add CelloSaaS model reference (CelloSaaS.Model) in your model project. Each model has to be inherited from "CelloSaaS.Model BaseEntity". Base Entity will add following properties to your business data model.

EntityIdentifier

Is used to model Identity. Make sure that you have added "EntityIdentifier" attribute in your business data model. This attribute is used to initialize EntityIdentifier property whenever you create an instance of the model.

The following code is an example for your business data model.

```
[EntityIdentifier(Name = "EmployeeDetails")]
    public class EmployeeDetails : BaseEntity
    {
        private string tenantId;
        private string firstName;
        private string lastName;
        private string middleName;
    }
```

Extended Row

It is used to have extended field details. Extended column values will be stored in the form of a key-value pair.

Identifier

It is used to have primary value of the record. You have to make sure to set and get Identifier while setting and fetching respectively. These properties are used to obtain extension features given by CelloSaaS. There should be a single Primary Key for the entities, and it is recommended to use GUID as the Primary Key.

DAL

Each model needs to have a corresponding CRUD DAL. For example, Employee model should have a corresponding `IEmployeeDAL<Employee>` and its data provider specific implementation for example `SqlDAL.EmployeeDAL` which inherits from this interface. Interface DAL should inherit from `CelloSaaS.DAL.IEntityDAL<T>` type T is the model type which in the case of example is Employee model and Implementation DAL should extend from `CelloSaaS.DAL.EntityDAL`.

Example

```
Public interface IEmployeeDAL : IEntityDAL<Employee>
Public class EmployeeDAL : EntityDAL, IEmployeeDAL
```

Service

In the services, to create an instance for DAL object, you need to use `DALImplementationFactory` and then call the operation with respective instance of `DataRequest`.

Example

```
IEmployeeDetailsDAL employeeDetailsDAL =
(IEmployeeDetailsDAL)DALImplementationFactory.GetDALImplementation(typeof(IEmployeeDetails
DAL));employeeDetailsDAL.Fetch(dataFeatchRequest);
```

When a business model object is fetched by calling `fetch` method, CelloSaaS will automatically fetch and bind the extended field details in your business model and when a business model object is inserted/updated/deleted the extension fields are automatically saved.

Database

Following are the steps to be followed in the data schema:

In most of the cases, business logic table will be considered as Entity or Business model object. To maintain the extended field values, you will have to create two tables for each entity. These table names and structures should be like the following.



If your business logic table name is Address, then you need to create two tables "AddressExtn"(Suffix with Extn) and "AddressExtnValue"(Suffix with ExtnValue) with the following structure (A script is attached with CelloSaaS package to create extension tables).
AddressExtnValue Structure

AddressExtnValue Structure

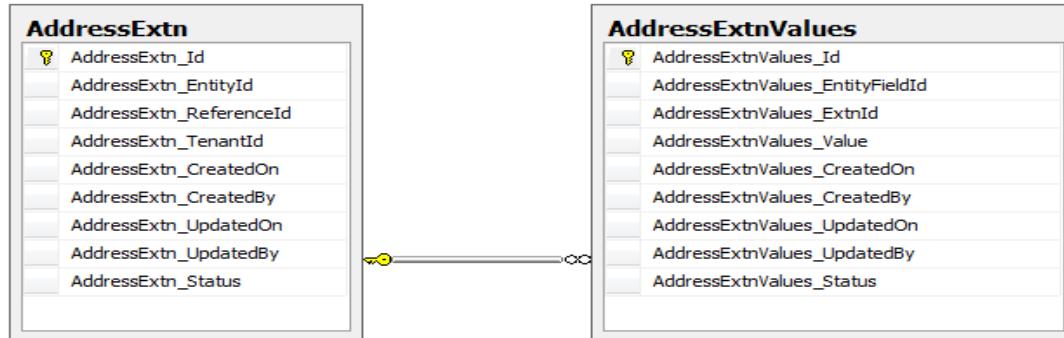


Figure 6-3– AddressExtnValues Structure Screen

Example

AddressExtn_Id	AddressExtn_EntityId	AddressExtn_ReferenceId	AddressExtn_TenantId
0016d359-5a01-42b6-8bc4-22431b16cb32	AddressDetails	0811fde4-65a4-4361-b418-77e7915a5787	TenantA

Example

AddressExtnValues_EntityFieldId	AddressExtnValues_ExtnId	AddressExtnValues_Value
Country	0016d359-5a01-42b6-8bc4-22431b16cb32	India

An entry for each entity with the following information needs to be placed in the Entity table.

- EntityIdentifier
- Schema Table name
- Extended schema table name
- Primary key of Schema table name

Make sure that EntityIdentifier is same as in EntityIdentifierAttribute in Model. Every Field in the system needs to have a unique ID in the system called “DataViewId” which is what is referred in the rest of the system to fetch the meta data of the corresponding field.

You will have to add Base Field details in EntityField table with the following information:

- EntityFieldIdentifier

- DataType
- FieldName

Atlast, Make sure that Entity is mapped with respective features. This has to be done in EntityFeature table.

Advantages

Single universal schema across all the tenants.

Flat Table Model

With this approach a few extra set of columns are defined for each data types supported by CelloSaaS and map these fields to the extended fields of the Tenants. This is a simple perspective of fixed data model. The Flat Table model implementation involves three tables. They are

EntityExtnColumnMetadata		EntityExtnColumnDetails	
EntityExtnColumnMetadata_Id		EntityExtnColumnDetails_Id	
EntityExtnColumnMetadata_EntityId		EntityExtnColumnDetails_EntityId	
DataFieldType_ID		EntityExtnColumnDetails_ColumnMetadataId	
EntityExtnColumnMetadata_NoOfColumn		EntityExtnColumnDetails_ColumnName	
EntityExtnColumnMetadata_Dimension		EntityExtnColumnDetails_CreatedBy	
EntityExtnColumnMetadata_CreatedBy		EntityExtnColumnDetails_CreatedOn	
EntityExtnColumnMetadata_CreatedOn		EntityExtnColumnDetails_UpdatedBy	
EntityExtnColumnMetadata_UpdatedBy		EntityExtnColumnDetails_UpdatedOn	
EntityExtnColumnMetadata_UpdatedOn		EntityExtnColumnDetails_Status	
EntityExtnColumnMetadata_Status			

Figure 6-4– Flat Table Model Tables Screen

EntityExtnColumnMetaDataTable

This table holds the metadata information about the Extended Fields . They are explained below

ID	EntityId	DataFieldType_ID	NoOfColumn	Dimension
Unique ID	Entity ID	Date Type	5	
5CC73BE2-5AD6-E111-B052-000000000000	Address	0CB87B97-1064-4BD5-9446-19F63CB44913	5	

Example

```
INSERT [dbo].[EntityExtnColumnMetadata] ( [EntityExtnColumnMetadata_EntityId],
[DataFieldType_ID], [EntityExtnColumnMetadata_NoOfColumn],
[EntityExtnColumnMetadata_Dimension], [EntityExtnColumnMetadata_CreatedBy],
[EntityExtnColumnMetadata_CreatedOn], [EntityExtnColumnMetadata_UpdatedBy],
[EntityExtnColumnMetadata_UpdatedOn], [EntityExtnColumnMetadata_Status]) VALUES
(N'Address', N'0cb87b97-1064-4bd5-9446-19f63cb44913', 5, N'0', N'Admin',
CAST(0x0000A09900674A07 AS DateTime), NULL, NULL, 1)
```

EntityExtnColumnDetails

The EntityExtnColumnDetails table contains “Alloted Extended fields column name” of each DataType(Int,Date,Varchar,PickUpField,Boolean,Bit) of each entity(Address,Tenant,..).

ID	EntityId	ColumnMetaData_ID	ColumnName
Unique ID	Entity ID	Extended column MetaData ID	Column1
5CC73BE2-5AD6-E111-B052-000000000000	Address	0CB87B97-1064-4BD5-9446-19F63CB44913	Column1

Example

```
INSERT [dbo].[EntityExtnColumnDetails] ( [EntityExtnColumnDetails_EntityId],
[EntityExtnColumnDetails_ColumnMetadataId], [EntityExtnColumnDetails_ColumnName],
[EntityExtnColumnDetails_CreatedBy], [EntityExtnColumnDetails_CreatedOn],
[EntityExtnColumnDetails_UpdatedBy], [EntityExtnColumnDetails_UpdatedOn],
[EntityExtnColumnDetails_Status]) VALUES ( N'Address', N'5CC73BE2-5AD6-E111-B052-000000000000', N'AddressExtn_Column1', N'Admin', CAST(0x0000A09A003D5D5A AS DateTime),
N'3398f837-b988-4708-999d-d3dfe11875b3', CAST(0x0000A09A00AF9E0F AS DateTime), 1)
```

<EntityName>Extn

This table will store the actual data. The TenantID field locks the data to that particular account or customer in the CelloSaaS multi-tenant system with a shared database. TableName represents the name of the table where the entity is stored. The EntityID is the primary key of whatever object this links to. The rest of the fields store actual data based on the type of the field. So whenever there is an insertion into **the extended field table**, our insert would look something like this.

When we pull the data from the base table, CelloSaaS dynamically appends the select script using Left join and pulls the tenant Specific Extended Fields from the Extended Fields table dynamically.

AddressExtn	
AddressExtn_Id	
AddressExtn_EntityId	
AddressExtn_ReferenceId	
AddressExtn_TenantId	
AddressExtn_Column1	
AddressExtn_Column2	
AddressExtn_Column3	
AddressExtn_Column4	
AddressExtn_Column20	
AddressExtn_Column21	
AddressExtn_Column22	
AddressExtn_Column23	
AddressExtn_Column24	

Figure 6-5– AddressExtn Screen

6.1.2 Configuration Settings

AppSettings.config

CelloSaaS supports 2 versions of extended fields, of which 1 can be ONLY be active. backward compatibility is provided in this version. To switch over between versions the following changes are to be made in the appsettings.config file

```
App Settings Entry
Key-Pair Table
<add key="ExtnTableFormat" value=" KeyValueTable">
Flat Table
<add key="ExtnTableFormat" value="FlatTable ">
```

```
Unity Config
<typeAlias alias="ExtendedSchemaDAL" type="CelloSaaS.SqlDAL.ExtendedSchemaDAL,
CelloSaaS.SqlDAL"/>
<typeAlias alias="EntityExtnSchemaDAL" type="CelloSaaS.SqlDAL.EntityExtnSchemaDAL,
CelloSaaS.SqlDAL"/>
Key-Pair Table
<type type="IExtendedSchemaDAL" mapTo="ExtendedSchemaDAL"
name="IExtendedSchemaDAL"></type>
Flat Table
<type type="IExtendedSchemaDAL" mapTo="EntityExtnSchemaDAL"
name="IExtendedSchemaDAL"></type>
```

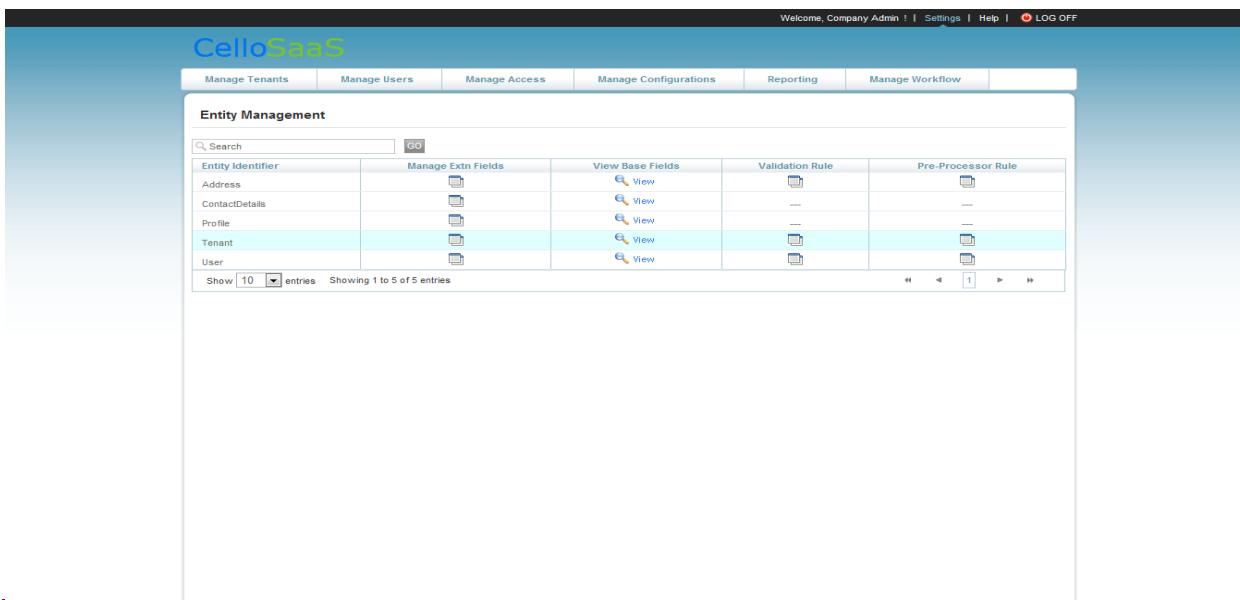
Migration from Key-Pair Model to Flat table Model

Refer the Script Included

Migration from Key-Pair Model to Flat table Model

Not supported

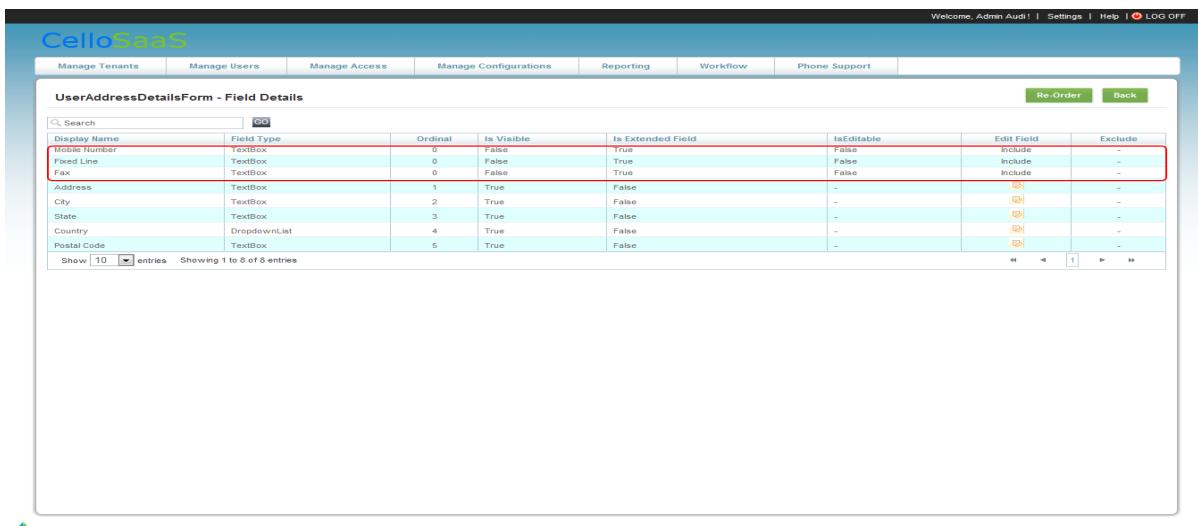
How to manage Extended Fields using User Interface?



The screenshot shows the Entity Management screen of the CelloSaaS application. At the top, there is a navigation bar with links: Manage Tenants, Manage Users, Manage Access, Manage Configurations, Reporting, and Manage Workflow. On the right side of the header, there are links for Welcome, Company Admin, Settings, Help, and LOG OFF. Below the header, the main title is "Entity Management". There is a search bar with a "GO" button. A table lists five entity types: Address, ContactDetails, Profile, Tenant, and User. Each row has columns for Entity Identifier, Manage Extn Fields, View Base Fields, Validation Rule, and Pre-Processor Rule. Each row also contains a "View" link under "View Base Fields". At the bottom of the table, there is a pagination control showing "Show 10 entries Showing 1 to 5 of 5 entries" and navigation arrows.

Figure 6-6– Entity Management Screen

The figure below displays the list of extension fields added to the Entity Employee Details. You can add, edit, and delete the extension fields. Click edit link to manage the configuration of the extension filed.



The screenshot shows the "UserAddressDetailsForm - Field Details" screen. At the top, there is a navigation bar with links: Manage Tenants, Manage Users, Manage Access, Manage Configurations, Reporting, Workflow, Phone Support, Re-Order, and Back. Below the navigation bar, the title is "UserAddressDetailsForm - Field Details". There is a search bar with a "GO" button. A table lists extension fields: Mobile Number, Fixed Line, Fax, Address, City, State, Country, and Postal Code. Each row has columns for Display Name, Field Type, Ordinal, Is Visible, Is Extended Field, IsEditable, Edit Field, and Exclude. The first three rows (Mobile Number, Fixed Line, Fax) are highlighted with a red border. At the bottom of the table, there is a pagination control showing "Show 10 entries Showing 1 to 8 of 8 entries" and navigation arrows.



Copyright © 2012 by techcello.com

Figure 6-7– Entity Management Screen

The figure below explains that configuration can be done for an extension field.



Welcome, Company Admin | Settings | Help | LOG OFF

CelloSaaS

Manage Tenants | Manage Users | Manage Access | Manage Configurations | Reporting | Workflow

DataView Field Properties

Display Name	Age	IsExtendedField	True
MaxLength	0	IsEditable	<input checked="" type="checkbox"/>
RegularExpression		IsMandatory	<input type="checkbox"/>
Field Type	TextBox	IsVisible	<input checked="" type="checkbox"/>
Description	A field for entering age.		

Update Back

Figure 6-8– DataView Field Properties Screen

Welcome, Company Admin | Settings | Help | LOG OFF

CelloSaaS

Manage Tenants | Manage Users | Manage Access | Manage Configurations | Reporting | Workflow

DataView Field Properties

Display Name	Address	IsExtendedField	False
Field Type	TextBox	IsVisible	<input checked="" type="checkbox"/>
Description	A field for entering address.		

Update Back



Copyright © 2012 by techcello.com

Figure 6-9 – DataView Field Properties Screen

Extended fields properties can be managed with the above screen. The properties are
Display name,

Data type
Length
Uniqueness
Regular expression

CelloSaaS supports following data types:

Varchar
Int
Float
Boolean
Date
Pickup Field

Each Data type has the following properties:

Properties	Description	Supporting Data Types
Length	Specify the size of the data.	Varchar.
Regular Expression	Specify the regular expression for data.	Varchar, Int, Float and Date.
Unique	Specify the data is unique or not.	Varchar, Int, Float, Date and Pickup Field.

Advantages

Relatively high performance
Retains the benefits of SQL technology

Searching Data from Extended Fields

Supported methods of connecting with Database:

ADO.NET
Entity Framework

Supported Data Types:

Varchar (string)
Int
Float
Boolean
Date

Primary key Data Types of the Entity Table:

Guid
Int
Varchar (string)

Supported Operators for extended fields searching:

ADO.NET	Entity Framework
And	And
Or	Or
Equals	Equals
NotEquals	NotEquals
GreaterThan	GreaterThan
LesserThan	LesserThan
GreaterThanOrEqual	GreaterThanOrEqual
LesserThanOrEqual	LesserThanOrEqual
Like	Like (Only for Varchar (string))
In	In Only for Varchar (string))
Between	Between

Make sure that the following settings are present in the following configuration file:

AppSettings.config

```
<!-- To turn OFF or ON the Extended Fields for the Entities-->
<add key="EnableExtendedFields" value="true"/>
<!-- Enable FlatTable -->
<add key="ExtnTableFormat" value="FlatTable"/>
```

Unity config

```
<TypeAliases>
<typeAlias alias="EntityExtnSchemaDAL" type="CelloSaaS.SqlDAL.EntityExtnSchemaDAL,
CelloSaaS.SqlDAL" />
</typeAliases>
<containers>
<container name="DataAccess">
<types>
<type type="IExtendedSchemaDAL" mapTo="ExtendedSchemaDAL"
name="IExtendedSchemaDAL"></type>
</types>
</container>
</containers>
```

Searching

If "Customer" is an entity, then search is performed in the "CustomerExtn" (EXTENSION TABLE) table and return the records that are matched.

Types of Searching

- Wildcard Search
- Regular Search

Wildcard Search

Search is made across all the extension fields or selective field of that entity and tenant by using the OR condition with the given search value.

Normal Search

Search is made on the extension field or fields of that entity and tenant with the given expression. The expression contains (operators (or, and, not, greater than, etc...), field name and its values).

API (ADO.NET)

This method returns the list of reference ids of the extension table that are matched with the given expression or search value

```
List<string> SearchExtendedFieldsReferencelds (ExtendedFieldsSearchCondition  
extendedFieldsSearchCondition);
```

This method returns the required query as string for getting reference ids of the extension table that are matched with the given expression or search value

```
string QueryForExtendedFieldsReferencelds (ExtendedFieldsSearchCondition  
extendedFieldsSearchCondition);
```

API (ENTITY FRAMEWORK)

This method returns the records of the entity table as object set, which are matched with reference ids of the extension table and primary key of the entity table with the given expression or search value with the sub query as input that returns reference ids of the extension table

```
static IQueryable<T> FilterCustomFields<T, E> (this IObjectSet<T> query, IQueryable<E>  
extendedEntity, string primaryKey, ExtendedFieldsSearchCondition condition)  
where T : BaseEntity  
where E : class
```

This method returns the records of the entity table as queryable, which are matched with reference ids of the extension table and primary key of the entity table with the given expression or search value with the sub query as input that returns reference ids of the extension table

```
static IQueryable<T> FilterCustomFields<T, E> (this IQueryable<T> query, IQueryable<E>  
extendedEntity, string primaryKey, ExtendedFieldsSearchCondition condition)  
where T : BaseEntity  
where E : class
```

This method returns the records of the entity table as object set, which are matched with reference ids of the extension table and primary key of the entity table with the given expression or search value with reference ids as input



```
static IQueryable<T> FilterCustomFieldsByReferencelds<T> (this IObjectSet<T> query,  
ExtendedFieldsSearchCondition extendedFieldsSearchCondition, string primaryKeyName,  
EFPByPrimaryKeyDataType dataType)  
where T: BaseEntity
```

This method returns the records of the entity table as queryable, which are matched with reference ids of the extension table and primary key of the entity table with the given expression or search value with reference ids as input

```
static IQueryable<T> FilterCustomFieldsByReferencelds<T>(this IQueryable<T> query,  
ExtendedFieldsSearchCondition extendedFieldsSearchCondition, string primaryKeyName,  
EFPByPrimaryKeyDataType dataType)  
where T: BaseEntity
```

TABLE AND MODEL CREATION

Consider an entity “Customer”.

Create the entity table with the entity name “Customer” and extension table as “CustomerExtn” (DBScripts are available for creating extension table).

For “Entity Framework” add the edmx file and then make “Code Generation Strategy” as “None”.

Model (class) for the entity table (“Customer”) must be created with same data type as of in that table.

Model must inherit BaseEntity.

Model must have attributes such as EntityIdentifier name(Name of the entity), EntityDescriptor (primary key name, DisplayName, SchemaTableName, ExtnSchemaTableName, IsExtensible, SchemaConnectionStringName)

Specify only name of the extension table and schema table. Do not include any schema name, database name.

If it is a wcf service call, then add the following service know type to the interface of extended field search.

```
[ServiceKnownTypeAttribute (typeof (Condition))]  
[ServiceKnownTypeAttribute (typeof (FieldCondition))]  
[ServiceKnownTypeAttribute (typeof (ValueCondition))]  
[ServiceKnownTypeAttribute (typeof (BracketCondition))]
```

Search Condition

For searching the entity with the extended fields you need the following class

ExtendedFieldsSearchCondition-Class from Cello Framework

```
/// <summary>  
/// Properties of Extended fields search condition  
/// </summary>  
[Serializable]  
[DataContract]  
public class ExtendedFieldsSearchCondition  
{  
    /// <summary>  
    /// Name of the entity
```



```
/// </summary>
[DataMember]
public string EntityIdentifier { get; set; }

/// <summary>
/// Tenant Id
/// </summary>
[DataMember]
public string TenantIdentifier { get; set; }

/// <summary>
/// Extended Field Value (This value is filled when the searching is going to be wildcard search)
/// </summary>
[DataMember]
public string SearchValue { get; set; }

/// <summary>
/// Extended fields search condition as model
/// </summary>
[DataMember]
public Condition SearchCondition { get; set; }

/// <summary>
/// final condition should be appended in this property
/// </summary>
[DataMember]
public string Condition { get; set; }

/// <summary>
/// SortFieldName(Field name)
/// </summary>
[DataMember]
public string SortFieldName { get; set; }

/// <summary>
/// SortFieldDirection(Ascending or descending)
/// </summary>
[DataMember]
public ExtendedFieldSorting SortDirection { get; set; }
}

/// <summary>
/// Interface that defines the property "Value"
/// </summary>
public interface ICondition
{
    /// <summary>
    /// Gets or sets the value
    /// </summary>
    [DataMember]
    string Value { get; set; }
}

/// <summary>
/// Properties of Extended fields search condition that inherits the interface "Icondition"
/// </summary>
[Serializable]
[DataContract]
```



```
public class Condition : ICondition
{
    /// <summary>
    /// Gets or sets the left hand side Operand(May be another expression or left hand side operand)
    /// </summary>
    [DataMember]
    public ICondition LHSOperand { get; set; }

    /// <summary>
    /// Gets or sets the right hand side Operand(May be another expression or right hand side operand)
    /// </summary>
    [DataMember]
    public ICondition RHSOperand { get; set; }

    /// <summary>
    /// Gets or sets the operator operates between the hand left hand side operand and the right hand
    /// side operand
    /// </summary>
    [DataMember]
    public ExtendedFieldsOperators? Operator { get; set; }

    /// <summary>
    /// Gets or sets the value
    /// </summary>
    [DataMember]
    public string Value{ get; set; }

    /// <summary>
    /// The class that inherits the interface "Icondition"
    /// </summary>
    [Serializable]
    [DataContract]
    public class FieldCondition : ICondition
    {
        /// <summary>
        /// Gets or sets the type of the data.
        /// </summary>
        /// <value>
        /// The type of the data.
        /// </value>
        [DataMember]
        public ExtendedFieldDataType DataType { get; set; }

        /// <summary>
        /// Gets or sets the value
        /// </summary>
        [DataMember]
        public string Value{ get; set; }

        /// <summary>
        /// The class that inherits the interface "Icondition"
        /// </summary>
        [Serializable]
        [DataContract]
        public class ValueCondition : ICondition
        {
            /// <summary>
            /// Gets or sets the value
            /// </summary>
```

```
/// </summary>
[DataMember]
public string Value{ get; set; }

/// <summary>
/// The class that inherit the class "condition" that includes bracket for the expression
/// </summary>
[Serializable]
[DataContract]
public class BracketCondition : Condition
{

}
```

Sorting is optional

If sorting is needed then we must provide sorting direction (ascending or descending) and field name to be sorted.

SortFieldName: Extended field name in which sorting is performed

SortDirection: Either ascending or descending

Creating Context for Entity Framework

```
public class CustomerEFContext : ObjectContext
{
    public CustomerEFContext()
        : base("name=CustomerEntities", "CustomerEntities")
    {
        this.ContextOptions.LazyLoadingEnabled = false;
    }

    public CustomerEFContext(EntityConnection conn)
        : base(conn)
    {
        this.ContextOptions.LazyLoadingEnabled = false;
    }

    public ObjectSet<Customer> Customers
    {
        get
        {
            if (_Customers == null)
            {
                _Customers = CreateObjectSet<Customer> ();
            }
            return _Customers;
        }
    }
    private ObjectSet<Customer> _Customers;

    public ObjectSet<CustomerExtn> CustomerExtns
    {
        get
        {
            if (_CustomerExtns == null)
            {
                _CustomerExtns = CreateObjectSet<CustomerExtn> ();
            }
        }
    }
}
```



```
    return _CustomerExtns;
}
}
private ObjectSet<CustomerExtn> _CustomerExtns;
}
```

Correct way: SchemaTableName = "clm.Customer", ExtnSchemaTableName = "CustomerExtn"
Incorrect way: SchemaTableName = "Customer", ExtnSchemaTableName = "clm.CustomerExtn"

Sample Model for Entity Framework

Entity Table Model

```
[EntityIdentifier (Name = "Customer")]
[EntityDescriptor (PrimaryKeyName = "Id", DisplayName = "CustomerNumber",
SchemaTableName = "clm.Customer", ExtnSchemaTableName = "CustomerExtn",
IsExtensible = true, SchemaTableConnectionStringName =
CelloSaaS.Library.DAL.Constants.ApplicationConnectionString)]
[Serializable]
[DataContract]
public class Customer : BaseEntity
{
[DataMember]
public Guid Id { get; set; }

[DataMember]
public Guid TenantId { get; set; }

[DataMember]
public int CustomerNumber { get; set; }

[DataMember]
public Guid PrimaryContactId { get; set; }

[DataMember]
public ContactDetail PrimaryContact { get; set; }

[DataMember]
public string CreatedBy { get; set; }

[DataMember]
public DateTime CreatedOn { get; set; }

[DataMember]
public string UpdatedBy { get; set; }

[DataMember]
public DateTime? UpdatedOn { get; set; }

[DataMember]
public bool Status { get; set; }
}
```

Extension Table Model



```
public class CustomerExtn
{
    public virtual System.Guid CustomerExtn_Id{get;set;}

    public virtual string CustomerExtn_EntityId{get;set;}

    public virtual System.Guid CustomerExtn_Referenceld{get;set;}

    public virtual System.Guid CustomerExtn_TenantId{get;set;}

    public virtual Nullable<int> CustomerExtn_Column1{get;set;}

    public virtual Nullable<int> CustomerExtn_Column2{get;set;}

    public virtual Nullable<int> CustomerExtn_Column3{get;set;}

    public virtual Nullable<int> CustomerExtn_Column4{get;set;}

    public virtual Nullable<int> CustomerExtn_Column5{get;set;}

    public virtual Nullable<System.DateTime> CustomerExtn_Column6{get;set;}

    public virtual Nullable<System.DateTime> CustomerExtn_Column7{get;set;}

    public virtual Nullable<System.DateTime> CustomerExtn_Column8{get;set;}

    public virtual Nullable<System.DateTime> CustomerExtn_Column9{get;set;}

    public virtual Nullable<System.DateTime> CustomerExtn_Column10{get;set;}

    public virtual string CustomerExtn_Column11{get;set;}

    public virtual string CustomerExtn_Column12{get;set;}

    public virtual string CustomerExtn_Column13{get;set;}

    public virtual string CustomerExtn_Column14{get;set;}

    public virtual string CustomerExtn_Column15{get;set;}

    public virtual string CustomerExtn_Column16{get;set;}

    public virtual string CustomerExtn_Column17{get;set;}

    public virtual string CustomerExtn_Column18{get;set;}

    public virtual string CustomerExtn_Column19{get;set;}

    public virtual string CustomerExtn_Column20{get;set;}

    public virtual string CustomerExtn_Column21{get;set;}

    public virtual string CustomerExtn_Column22{get;set;}

    public virtual string CustomerExtn_Column23{get;set;}

    public virtual string CustomerExtn_Column24{get;set;}
```



```
public virtual string CustomerExtn_Column25{get;set;}  
public virtual Nullable<double> CustomerExtn_Column26{get;set;}  
public virtual Nullable<double> CustomerExtn_Column27{get;set;}  
public virtual Nullable<double> CustomerExtn_Column28{get;set;}  
public virtual Nullable<double> CustomerExtn_Column29{get;set;}  
public virtual Nullable<bool> CustomerExtn_Column31{get;set;}  
public virtual Nullable<bool> CustomerExtn_Column32{get;set;}  
public virtual Nullable<bool> CustomerExtn_Column33{get;set;}  
public virtual Nullable<bool> CustomerExtn_Column34{get;set;}  
public virtual Nullable<bool> CustomerExtn_Column35{ get; set; }  
public virtual System.DateTime CustomerExtn_CreatedOn{ get; set; }  
public virtual string CustomerExtn_CreatedBy{ get; set; }  
public virtual Nullable<System.DateTime> CustomerExtn_UpdatedOn{ get; set; }  
public virtual string CustomerExtn_UpdatedBy{ get; set; }  
public virtual bool CustomerExtn_Status{ get; set; }
```

Sample Class for using Extended Field Search

This class must inherit **SearchCondition** that should be injected with **ExtendedFieldsSearchCondition** class

```
[Serializable]  
[DataContract]  
public class CustomerSearchCondition : SearchCondition  
{  
    [DataMember]  
    public string[] Identifiers { get; set; }  
  
    [DataMember]  
    public string[] TenantIds { get; set; }  
  
    [DataMember]  
    public int CustomerNumber { get; set; }  
  
    [DataMember]  
    public string Condition{ get; set; }  
  
    [DataMember]  
    public ExtendedFieldsSearchCondition ExtendedFieldsSearchCondition { get; set; }  
}
```

Example Usage

Utilizing Wild Card Search in Controller

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult CustomerList(FormCollection formCollection)
{
    CustomerSearchCondition customerSearchCondition = new CustomerSearchCondition();
    customerSearchCondition.ExtendedFieldsSearchCondition = new
    ExtendedFieldsSearchCondition();
    customerSearchCondition.ExtendedFieldsSearchCondition.EntityIdentifier = "Customer";
    customerSearchCondition.ExtendedFieldsSearchCondition.TenantIdentifier = this.TenantId;
    customerSearchCondition.ExtendedFieldsSearchCondition.SearchValue = searchValue;
    var list = CustomerServiceProxy.SearchCustomerDetails(customerSearchCondition);
}
```

Sample Query

Select * from CustomerExtn where CustomerExtn_Column1 like '2' or CustomerExtn_Column2 like '3' AND CustomerExtn_EntityId='Customer' AND CustomerExtn_TenantId='B590CD25-3093-DF11-8DEB-001EC9DAB123'

Regular Search

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult CustomerList(FormCollection formCollection)
{
    // (FirstName LIKE '%Sam%' OR (ExtnNumber>=1 AND ExtnNumber<=100)) AND
    // (EmployeeNumber IN (1,2,3) AND (EmailId LIKE '%Sam@techcello.com%'))
    CustomerSearchCondition customerSearchCondition = new CustomerSearchCondition();
    ExtendedFieldsSearchCondition extendedfieldsSearchCondition = new
    ExtendedFieldsSearchCondition();
    extendedfieldsSearchCondition.EntityIdentifier = "Tenant";
    extendedfieldsSearchCondition.TenantIdentifier = "B590CD25-3093-DF11-8DEB-001EC9DAB123";
    extendedfieldsSearchCondition.SearchCondition = new Condition();
    var C1 = new BracketCondition();
    C1.LHSOperand = new FieldCondition();
    C1.LHSOperand.DataType= ExtendedFieldDataType.Varchar;
    C1.LHSOperand.Value = "FirstName";
    C1.RHSOperand = new ValueCondition();
    C1.RHSOperand.Value = "Sam";
    C1.Operator = ExtendedFieldsOperators.Like;
    var C2 = new BracketCondition();
    C2.LHSOperand = new FieldCondition();
    C2.LHSOperand.DataType= ExtendedFieldDataType.Int;
    C2.LHSOperand.Value = "ExtnNumber";
    C2.RHSOperand = new ValueCondition();
    C2.RHSOperand.Value = "1,100";
    C2.Operator = ExtendedFieldsOperators.Between;
    var C3 = new Condition();
    C3.LHSOperand = new FieldCondition();
    C3.LHSOperand.DataType= ExtendedFieldDataType.Int;
    C3.LHSOperand.Value = "EmployeeNumber ";
    C3.RHSOperand = new ValueCondition();
    C3.RHSOperand.Value = "1,2,3";
    C3.Operator = ExtendedFieldsOperators.In;
    var C4 = new Condition();
    C4.LHSOperand = new FieldCondition();
    C4.LHSOperand.DataType= ExtendedFieldDataType.Varchar;
```

```

C4.LHSOperand.Value = "EmailId";
C4.RHSOperand = new ValueCondition();
C4.RHSOperand.Value = "Sam@techcello.com";
C4.Operator = ExtendedFieldsOperators.Like;
var C5 = new Condition();
C5.LHSOperand = C1;
C5.RHSOperand = C2;
C5.Operator = ExtendedFieldsOperators.Or;
var C6 = new Condition();
C6.LHSOperand = C3;
C6.RHSOperand = C4;
C6.Operator = ExtendedFieldsOperators.And;
var C7 = new Condition();
C7.LHSOperand = C5;
C7.RHSOperand = C6;
C7.Operator = ExtendedFieldsOperators.And;
extendedfieldsSearchCondition.SearchCondition = C7;
customerSearchCondition.ExtendedFieldsSearchCondition = extendedfieldsSearchCondition;
customerSearchCondition.ExtendedFieldsSearchCondition.EntityIdentifier = "Customer";
customerSearchCondition.ExtendedFieldsSearchCondition.TenantIdentifier = this.TenantId;
var list = CustomerServiceProxy.SearchCustomerDetails(customerSearchCondition);
}

```

Final Query

Select * from CustomerExtn where (CustomerExtn_Column1 like '%Sam%' or (CustomerExtn_Column2 like '%Daniel%')) AND CustomerExtn_Column3 like '%Sam%' or (CustomerExtn_Column4 like '%Daniel %' AND CustomerExtn_EntityId='Customer' AND CustomerExtn_TenantId='B590CD25-3093-DF11-8DEB-001EC9DAB123'

DAL

ADO.NET

Search with sub query that returns reference IDs

```

protected override Dictionary<string, Customer> DoSearch(DataSearchRequest
dataSearchRequest)
{
var condition = dataSearchRequest.SearchCondition as CustomerSearchCondition;
var fetchQuery = new FetchQuery(new Customer()).EntityIdentifier, FetchType.View,
GetConnectionStringName(), dataSearchRequest.OperationalTenantId);
fetchQuery.SelectCondition = new StringBuilder("SELECT * FROM [Customer] AS c");
var filterQuery = new StringBuilder(" (c.Status=1 ");
fetchQuery.FilterCondition = filterQuery;
if (condition.Identifiers != null && condition.Identifiers.Length > 0 )
{
if (condition.Identifiers.Length == 1)
{
filterQuery.Append(" AND c.Id = @Id");
fetchQuery.AddParameter("@Id", condition.Identifiers[0]);
}
else
{
filterQuery.Append(" AND c.Id IN (" + string.Join(", ", condition.Identifiers) + ")");
}
}

```

```

else if (condition.CustomerNumber > 0 &&
!string.IsNullOrEmpty(condition.ExtendedFieldsSearchCondition.Condition))
{
filterQuery.Append(" AND (c.CustomerNumber=" + condition.CustomerNumber + "OR c.Id IN (" +
condition.ExtendedFieldsSearchCondition.Condition + "))");
}
filterQuery.Append(")");
var dbAccess = new DBAccess(GetConnectionStringName());
IDataReader reader = null;
try
{
Context.ElevatePrivilege();
reader = dbAccess.ExecuteReaderWithoutTenantContext(fetchQuery);
Dictionary<string, Customer> entities = new Dictionary<string, Customer>();
while (reader != null && reader.Read())
{
var entity = GetEntityFromReader(reader);
entities.Add(entity.Identifier, entity);
}
return entities;
}
finally
{
reader.CloseReader();
Context.ElevatePrivilege();
}
}

```

Search with list of Reference IDs

```

protected override Dictionary<string, Customer> DoSearch(DataSearchRequest
dataSearchRequest)
{
var condition = dataSearchRequest.SearchCondition as CustomerSearchCondition;
var extendedFieldDAL = DALImplementationFactory.Resolve<IExtendedFieldsSearchService>();
List<string> referenceIds =
extendedFieldDAL.SearchExtendedFieldsReferenceIds(condition.ExtendedFieldsSearchCondition);
var fetchQuery = new FetchQuery(new Customer().EntityIdentifier, FetchType.View,
GetConnectionStringName(), dataSearchRequest.OperationalTenantId);
fetchQuery.SelectCondition = new StringBuilder("SELECT * FROM [Customer] AS c");
var filterQuery = new StringBuilder(" (c.Status=1 ");
fetchQuery.FilterCondition = filterQuery;
if (condition.Identifiers != null && condition.Identifiers.Length > 0 )
{
if (condition.Identifiers.Length == 1)
{
filterQuery.Append(" AND c.Id = @Id");
fetchQuery.AddParameter("@Id", condition.Identifiers[0]);
}
else
{
filterQuery.Append(" AND c.Id IN (" + string.Join(", ", condition.Identifiers) + ")");
}
}
else if (condition.CustomerNumber > 0 &&
!string.IsNullOrEmpty(condition.ExtendedFieldsSearchCondition.Condition
))
{

```

```

filterQuery.Append(" AND (c.CustomerNumber=" + condition.CustomerNumber + "OR c.Id IN (" + 
string.Join(",", referencelds) + "))");
}
filterQuery.Append(")");
var dbAccess = new DBAccess(GetConnectionStringName());
IDataReader reader = null;
try
{
Context.ElevatePrivilege();
reader = dbAccess.ExecuteReaderWithoutTenantContext(fetchQuery);
Dictionary<string, Customer> entities = new Dictionary<string, Customer>();
while (reader != null && reader.Read())
{
var entity = GetEntityFromReader(reader);
entities.Add(entity.Identifier, entity);
}
return entities;
}
finally
{
reader.CloseReader();
Context.ElevatePrivilege();
}
}

```

Entity Framework

Search with SubQuery that returns Reference IDs

```

protected override Dictionary<string, Customer> DoSearch(DataSearchRequest
dataSearchRequest)
{
using (CustomerEFContext context = new CustomerEFContext(GetConnectionObject()))
{
context.Customers.MergeOption = System.Data.Objects.MergeOption.NoTracking;
context.ContextOptions.LazyLoadingEnabled = false;
CustomerSearchCondition condition = dataSearchRequest.SearchCondition as
CustomerSearchCondition;
var query = context.Customers.Where(c => c.Status == true);
if (condition != null && condition.ExtendedFieldsSearchCondition != null)
{
query = query.FilterCustomFields(context.CustomerExtns, "Id",
condition.ExtendedFieldsSearchCondition).Where(c => c.Status == true);
}
if (condition.TenantIds != null && condition.TenantIds.Length > 0)
{
var tids = condition.TenantIds.Select(x => Guid.Parse(x));
query = query.Where(c => tids.Contains(c.TenantId));
}
if (dataSearchRequest != null && dataSearchRequest.Identifiers != null)
{
var cids = dataSearchRequest.Identifiers.Select(x => Guid.Parse(x));
query = query.Where(c => cids.Contains(c.Id));
}
else
{
if (condition.CustomerNumber > 0)
{

```



```
query = query.Where(c => c.CustomerNumber == condition.CustomerNumber);
}
}
var customerDetails = new Dictionary<string, Customer>();
foreach (var cus in query)
{
cus.Identifier = cus.Id.ToString();
customerDetails.Add(cus.Identifier, cus);
}
return customerDetails;
}
}
```

Search with list of Reference IDs

```
protected override Dictionary<string, Customer> DoSearch(DataSearchRequest
dataSearchRequest)
{
using (CustomerEFContext context = new CustomerEFContext(GetConnectionObject()))
{
context.Customers.MergeOption = System.Data.Objects.MergeOption.NoTracking;
context.ContextOptions.LazyLoadingEnabled = false;
CustomerSearchCondition condition = dataSearchRequest.SearchCondition as
CustomerSearchCondition;
var query = context.Customers.Where(c => c.Status == true);
if (condition != null && condition.ExtendedFieldsSearchCondition != null)
{
query = query.FilterCustomFieldsByReferencelds(condition.ExtendedFieldsSearchCondition, "Id",
EFPrimaryKeyType.GUID).Where(c => c.Status == true);
}
if (condition.TenantIds != null && condition.TenantIds.Length > 0)
{
var tids = condition.TenantIds.Select(x => Guid.Parse(x));
query = query.Where(c => tids.Contains(c.TenantId));
}
if (dataSearchRequest != null && dataSearchRequest.Identifiers != null)
{
var cids = dataSearchRequest.Identifiers.Select(x => Guid.Parse(x));
query = query.Where(c => cids.Contains(c.Id));
}
else
{
if (condition.CustomerNumber > 0)
{
query = query.Where(c => c.CustomerNumber == condition.CustomerNumber);
}
}
var customerDetails = new Dictionary<string, Customer>();
foreach (var cus in query)
{
cus.Identifier = cus.Id.ToString();
customerDetails.Add(cus.Identifier, cus);
}
return customerDetails;
}
}
```

How To Guide: [Implementing Extension Fields](#)

6.2 Settings Template

The configurable tenant settings such as themes, logo, date and time and locale etc within the system can be packaged as templates for easier management. A tenant configuration is a set of attributes which can be assigned with values and can be modified later. The value could be any .net primitive data types such as *bool, string, datetime, int, and decimal*. It works in a many-to-one relationship i.e. multiple templates can be assigned to a single tenant. The configuration settings then possessed by the tenant is a collation of the settings from all their assigned templates. When creating sub tenant, the administrator can then group one or more attributes from the global template and assign it to the Sub tenant.

A template can be either a fixed or customizable template or organization specific template. When a template is fixed and assigned to a tenant, then the settings within the template cannot be altered by the tenant and it cannot be used for their sub tenants. When a template is customizable and when it is assigned to a tenant the settings within the template can be modified by them and the template can further be used to assign to the child tenants. Any tenant who has the privilege to create templates would be able to create templates and templates created by Administrator will be cascaded to all the tenants and sub tenants which can be used by them.

To Create a custom template, follow the below step

Login to the admin portal

Go to Manage Configuration → Tenant Settings → Manage Settings Template

Upon selecting any tenant in the Tenant list dropdown will display a grid with the list of already created templates

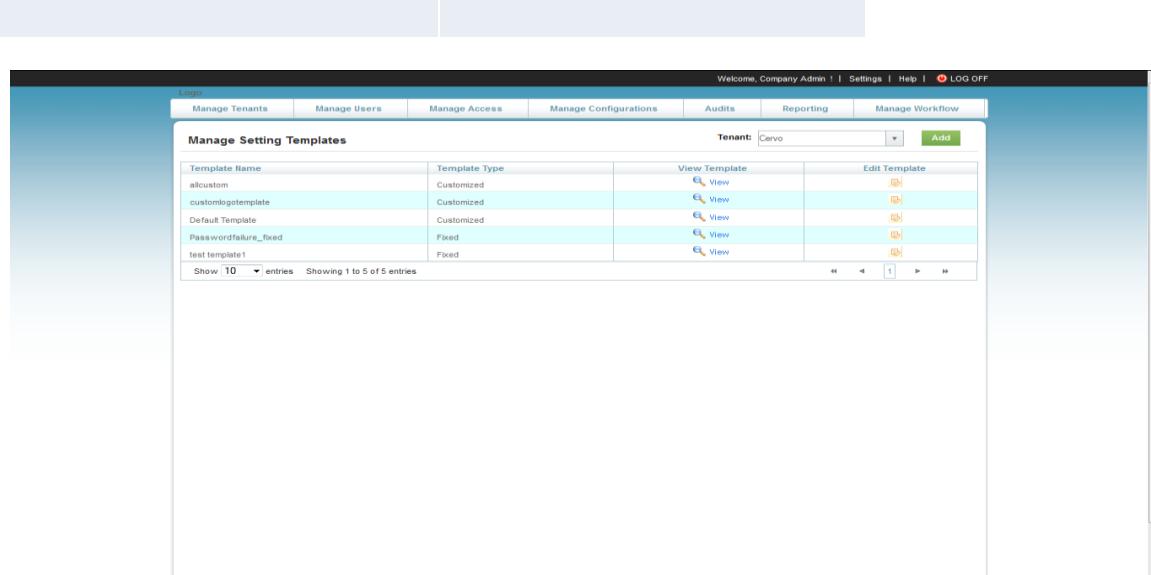
Click on Add button, it displays tenant settings screens

Fill the required attributes and click on Save to store it in Database

The Attributes are below:

Properties	Description
Name	Name of the Template (Unique)
Type	Fixed or Customized
IS Global	If checked, the Template will be available globally for all the Tenants
WCF Shared Key	An Encrypted string used as key to uniquely identify the WCF Calls
User Password Expiration Days	No of days to expire user password
Application Connection String	The Tenant wise Application DB Connection String
Home Realm	Home Realm for Single Sign On (SSO)

Link By Other Tenant's Users	Enable/Disable Link By Other Tenant's Users
Maximum Password Answer Failure Count	Maximum Password Answer Failure Count
Maximum Password Failure Count	Maximum Password Failure Count
User Connection String	Connection string of Authentication Store
Auto Approval For Tenant Creation	Enable to Auto Approve Tenant Creation
Theme	Select a theme to apply for a Tenant
Logo	Logo of the Tenant
Date Format String	Date Format String
Enable Product Analytics	Enable to capture Product Analytics
Language	Setting to select a Language for a Tenant



Template Name	Template Type	View Template	Edit Template
alcustom	Customized	View	Edit
customlogotemplate	Customized	View	Edit
Default Template	Customized	View	Edit
Passwordfailure_fixed	Fixed	View	Edit
test template1	Fixed	View	Edit

Show 10 ▾ entries Showing 1 to 5 of 5 entries

Figure 6-10– Manage Settings Templates Screen

The screenshot shows the 'Manage Template' page. At the top, there's a header with navigation links: 'Welcome, Company Admin 1 | Settings | Help | LOG OFF'. Below the header is a 'Template Details' section with fields for 'Name' (with a required asterisk), 'Type' (radio buttons for 'Fixed' or 'Customized'), and 'Is Global' (checkbox). A note says 'Please select the below attributes, then provide a value'. Below this is a table with columns 'Select' and 'Attribute Name'. The table lists various attributes like 'WCF Shared Key', 'Tenant Authentication Setting', etc., each with a corresponding 'Attribute Value' input field. Some values have a small note like 'ConnectionString-Provider'.

Figure 6-11– Manage Templates Screen

The List displays the global templates (Created by Product Admin) and the logged in tenant's own templates. The List has the options to view and edit the template.

The screenshot shows the 'Manage Setting Templates' page. At the top, there's a header with navigation links: 'Welcome, Company Admin 1 | Settings | Help | LOG OFF'. Below the header is a table titled 'Manage Setting Templates' with columns 'Template Name', 'Template Type', and 'View Template'. The table contains five entries: 'allcustom' (Customized), 'customlogotemplate' (Customized), 'Default Template' (Customized), 'Passwordfailure_fixed' (Fixed), and 'test template1' (Fixed). To the right of the table is a 'Tenant' dropdown menu. The dropdown is currently set to 'Ritz' and shows a list of other tenants: 'Ritz', 'Etiga', 'Cervo', 'Alto', 'Swift VDI', 'Benz', 'hundai', 'Tata', 'audi', 'wowwhattenant', 'Swift', 'SwiftDzire', 'Company', 'Maruthi', 'Ford', and 'Skoda'. There are also buttons for 'Add' and 'Show All Tenants'.

Figure 6-12– Manage Settings Templates Screen

Services

```
/// <summary>
/// Gets all settings templates.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
Dictionary<string,SettingsTemplate> GetSettingsTemplatesForTenant(string tenantId);
```

Add Settings Template

The Add Settings Template is used to add/create the new Settings Template. In this, we have to give unique *Template Name*, *Template Type*, and *Settings Attribute*. The selected settings attribute

values will be given in their appropriate textbox, dropdown or input controls. The AddSettingsTemplate method adds the Settings Template for the logged in user tenant.

```
/// <summary>
/// Adds the settings template.
/// </summary>
/// <param name="settingsTemplate">The settings template.</param>
/// <returns></returns>
string AddSettingsTemplate(SettingsTemplate settingsTemplate);
```

Edit Settings Template

The template is completely extended by adding or removing or changing the attributes of the template. If a Tenant/User adds an attribute to the template, the framework will check the existence of any of the other tenant's templates contains the same attribute. If so the logged in user will not be allowed to add the same attribute to the template. If the user removes the attribute from the template, the system will check the tenant settings accordingly and removes the corresponding attribute from the tenant's settings.

```
/// <summary>
/// Updates the settings template.
/// </summary>
/// <param name="settingsTemplate">The settings template.</param>
void UpdateSettingsTemplate(SettingsTemplate settingsTemplate);
```

The UpdateSettingsTemplate method is used to update the settings template and attributes of the settings template.

View Settings Template

It is used to get the template and its attribute details. The attribute details contain the attribute name and the corresponding attribute value.

```
/// <summary>
/// Gets all settings templates.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
Dictionary<string, SettingsTemplate> GetSettingsTemplatesForTenant(string tenantId);
```

Tenant Settings Template

This is used to map the template with the tenants and sub tenants, the already mapped templates will also be listed out along with the mapping functionality. Tenant Admin can view, edit, and remove the template mapping for the tenant.

Tenant Settings Template List

It lists out all the Tenant Settings Template and the sub tenants template settings.

```
/// <summary>
/// Gets the tenant settings templates.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
Dictionary<string, TenantSettingsTemplate> GetTenantSettingsTemplates(string tenantId);
```

Map Tenant Settings Template

It is used to map or set the template and its settings to the tenant. Here, the template which can be assignable for the child tenant will be shown in the dropdown list. The tenant will choose the assignable template and the tenant can save the template to their child tenant.

```
/// <summary>
/// Gets the assignable template for tenant.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
Dictionary<string, SettingsTemplate> GetAssignableTemplateForTenant(string tenantId);

/// <summary>
/// Adds the tenant settings template.
/// </summary>
/// <param name="tenantSettingsTemplate">The tenant settings template.</param>
/// <returns></returns>
string AddTenantSettingsTemplate(TenantSettingsTemplate tenantSettingsTemplate);
```

Edit tenant Settings Template

The logged in Tenant can edit the template which is mapped to his tenant account. Here the tenant can only able to change the values of the settings attribute of the template.

```
/// <summary>
/// Updates the tenant settings template.
/// </summary>
/// <param name="tenantSettingsTemplate">The tenant settings template.</param>
void UpdateTenantSettingsTemplate(TenantSettingsTemplate tenantSettingsTemplate);
```

Delete Tenant Settings Template

This will delete the template which is mapped with the tenant and removes the corresponding settings attribute from the tenant settings of the tenant.

```
/// <summary>
/// Deletes the tenant template.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <param name="templateId">The template id.</param>
void DeleteTenantSettingsTemplate(string tenantId, string templateId);
```

How To Guide: [Logo & Themes Customization](#)

6.3 Forms & Grids Customization

Like “Data Model Extension”, the SaaS product should provide their tenants the capability to customize the fields in forms and grids. Tenants and end users may want to see different

information in the grids. CelloSaaS offers Forms and Grids Customization features to help the developers build this customization support to their products easily. CelloSaaS supports the following:

- Modification of core field display name per tenant
- Switching off core fields per tenant
- Adding Fields extended via DataField Extension to the forms and Grids
- Managing the visibility, edit ability, validation of the extension fields
- Reposition of columns within the form and grid

Database

As a first step, you need to define a DataView in the database which is used to manage the view of business model entity. DataView can be of two types, it could be either Form or Grid. Like entity fields table, you will have to enter some back end data entries for base fields.

DataView	
Key	DataView_ID
	DataView_DataViewID
	DataView_Name
	DataView_Application
	DataView_Description
	DataView_MainEntity

Figure 6-13– DataView Screen

Every DataView must be mapped with an entity.

Example

DataViewId	DataViewName	DataViewDescription	MainEntity
FormManageEmployee	BasicDetails	BasicDetails	EmployeeDetails

Every Data View be it form, grid needs to have an unique ID in the system called “DataViewId”. This is what is referred in the rest of the system to fetch the meta data of the corresponding forms or grids.

Enter the base field entries in DataViewFields table with the following information.

Column Name	Data Type	Allow Nulls

EntityFieldIdentifier	Varchar (50)	False
DataViewId	Integer	True
TypeId	Integer	False
ExtendedTenantCode	Varchar (50)	True

Example

EntityFieldIdentifier	DataViewId	TypeId	ExtendedTenantCode
EmployeeId	1	1	TenantA

You will have to set the properties of these fields in DataViewFieldProperties table with the following information.

DataViewFields	
<input checked="" type="checkbox"/>	DataViewField_ID
<input type="checkbox"/>	DataViewField_FieldID
<input type="checkbox"/>	DataViewField_DataViewID
<input type="checkbox"/>	DataViewField_TypeID
<input type="checkbox"/>	DataViewField_ExtendedTenantCode
<input type="checkbox"/>	DataViewField_GlobalId
<input type="checkbox"/>	DataViewField_IsGlobalField
<input type="checkbox"/>	DataViewField_IsFixed

Figure 6-14 – DataViewFields Screen

Example

DataViewFieldId	Display Name	Ordinal	IsVisible	IsEditable	IsMandatory	Reg Expression	Max Length	Pick ListId	IsMultiLine
EmployeeId	EmployeeId	1	1	1	1	Null	10	Null	Null

6.3.1 Form Customization

View

CelloSaaS provides a component called CelloSaaS.View.DataViewTable which needs to be used to render forms. This is available in the assembly CelloSaaS.View.

To enable forms customization, developers should use a class called CelloSaaS.View.DataViewTable to construct the table. The structure of CelloSaaS.View.DataViewTable is as follows.

```
CelloSaaS.View.DataViewTable(string name, CurrentFormMode formMode, Dictionary<string, string> formAttributes, HtmlHelper htmlHelper, int noOfCols, Dictionary<int, ColumnProperties> columnProperties, AlternateRowStyle alternateRowStyle, BaseEntity baseEntity, string modelPrefix)
```

Name - Dataview name

FormMode - Current mode of the form (View,Edit,Insert)

FormAttributes - Html attributes for the form table

HtmlHelper - Html object of the current page

NoofCols - Number of column for the table

ColumnProperties - Properties of the column in key-value collection.Key-Column no,Value-Column Properties

AlternateRowStyle - Row style for the table

BaseEntity - Entity object

ModelPrefix - ModelPrefix which is used to bind extended rows

```
///<summary>
/// Creates new table cell content for insert section
///</summary>
void StartInsertSection()
///<summary>
/// Assigns the passed table cells in Insert section
///</summary>
void EndInsertSection()
///<summary>
/// Creates new table cell content for Edit section
///</summary>
void StartEditSection()
///<summary>
/// Assigns the passed table cells in Edit section
///</summary>
void EndEditSection()
///<summary>
/// Creates new table cell content for view section
///</summary>
void StartViewSection()
///<summary>
/// Assigns the passed table cells in View section
///</summary>
void EndViewSection()
///<summary>
/// Used to create a new table with new cell content
///</summary>
///<param name="containingFormFieldIdentifier">Unique Identifier for the field</param>
void StartTableCell(string containingFormFieldIdentifier)
///<summary>
/// Adds field label and html attributes
///</summary>
///<param name="attributesString">Html attributes for the label</param>
///<param name="containingFormFieldIdentifier">Unique Identifier for the field</param>
///<param name="value">Name which will be displayed</param>
void AddFieldLabel(string attributesString, string containingFormFieldIdentifier, string value)
```

```

///<summary>
/// Adds the content of the cell in form with passed fieldIdentifier
///</summary>
///<param name="containingFormFieldIdentifier">Unique Identifier for the field</param>
///<param name="content">Content of the cell</param>
void AddFieldContent(string containingFormFieldIdentifier, string content)
///<summary>
/// Renders the form table with the information passed as constructor parameters and added cell
///</summary>

```

Example

```

DataViewTable frmTable = new CelloSaaS.View.DataViewTable("FormAddEmployee"
currentMode,
formAttributes,
this.Html,
1,
columnAttributes,
alternateRowStyle,
objEmployeeForm,
"EmployeeDetails");

```

FormAddEmployee – Dataview Id which is added in dataview table

CurrentMode – Insert, Edit or Delete

FormAttributes - Defines html attributes for the form table

Example

```

Dictionary<string
string> formAttributes = new Dictionary<string>();
formAttributes.Add("id""frmAdd");
formAttributes.Add("class""profile-det-container-inner-det1");
formAttributes.Add("border""0");
formAttributes.Add("cellspacing""1");
formAttributes.Add("cellpadding"
"0");

```

Example

```

Dictionary<int, ColumnProperties> columnAttributes = new Dictionary<int, ColumnProperties>();
columnAttribute = new ColumnProperties();
columnAttribute.LabelContentProperties = new Dictionary<string, string>() {
    { "class", "aln-rht" } };
columnAttribute.FieldContentProperties = new Dictionary<string, string>() { };
columnAttributes.Add(1, columnAttribute);

```

AlternateRowStyle – Defines style for the alternate row

ObjEmployeeForm – It is actual business model to be bound in form table

EmployeeDetails – Shows the prefix of the model

You need to define your forms with three templates of Insert, Edit and View. The template code will be similar to the one below.

```

//Insert template
frmTable.StartInsertSection();
frmTable.StartTableCell("EmployeeLastName");
frmTable.AddFieldLabel("", "EmployeeLastName", "LastName");

```

```

content = newStringBuilder();
content.Append(Html.TextBox("EmployeeDetails.LastName",
objEmployeeForm.LastName, new { maxlength = 200 }));
content.Append(Html.ValidationMessage("EmployeeDetails.LastName", "*"));
frmTable.AddFieldContent("EmployeeLastName", content.ToString());
frmTable.EndInsertSection();
//Edit template
frmTable.StartEditSection();
content = newStringBuilder();           frmTable.StartTableCell("EmployeeLastName");
frmTable.AddFieldLabel("", "EmployeeLastName", "LastName");
content.Append(Html.TextBox("EmployeeDetails.LastName",
objEmployeeForm.LastName, new { maxlength = 200 }));
content.Append(Html.ValidationMessage("EmployeeDetails.LastName", "*"));
frmTable.AddFieldContent("EmployeeLastName", content.ToString());
frmTable.EndEditSection();
//View template
frmTable.StartViewSection();
content = newStringBuilder();
frmTable.StartTableCell("EmployeeLastName");
frmTable.AddFieldLabel("", "EmployeeLastName", "LastName");
content.Append(Html.Hidden("EmployeeDetails.LastName", objEmployeeForm.LastName));
content.Append(Html.Encode(objEmployeeForm.LastName));
content.Append(Html.ValidationMessage("EmployeeDetails.LastName", "*"));
frmTable.AddFieldContent("EmployeeLastName", content.ToString());
frmTable.EndViewSection();
string table = frmTable.Render();
Response.Write(table);

```

Here, you will notice that all the fields are added in the formtable along with what CelloSaaS calls “Field Identifiers.

Example

```
frmTable.AddFieldContent("EmployeeLastName" content.ToString());
```

In this, **"EmployeeLastName"** is the field identifier. Every field in the system needs to have a unique identifier, and this is the link between the field and its configurations. This is how you will have to use our CelloSaaS DataViewTable component. Managing extension fields will be taken care by DataViewTable. CelloSaaS.DataViewTable does support field level access with data scope. Refer Field level access feature for more information.

When a model is updated, the extended field values are bound implicitly with the business model object using CelloSaaS.View.ExtendedEntityRowBinder which overrides MVC's DefaultModelBinder. To enable this, you will have to register this binder along with DefaultModelBinder by adding the following code in your Global.asax.cs file.

```

protected void Application_Start()
{
    RegisterRoutes(RouteTable.Routes);
    RegisterBinders();
}
private void RegisterBinders()
{
    ModelBinders.Binders.DefaultBinder = new
    CelloSaaS.View.BusinessModelBinder();
    ModelBinders.Binders.Add(typeof(CelloSaaS.Model.DataManagement.ExtendedEntityRow), new
    CelloSaaS.
    .View.ExtendedEntityRowBinder());
}

```

6.3.2 Grid Customization

To implement Grid customization, CelloSaaS provides a component called CelloSaaS.View.celloGrid. You need to add CelloSaaS reference (CelloSaaS.View) in your MVC application. CelloSaaS uses MvcContrib Grid to construct HTML tables for displaying data from a collection of business model object. celloGrid also supports field level access with datascope.

To know more about this, refer it in Field level access feature. Implementation of celloGrid is very similar as we do with MvcContrib grid.

To enable Grid customization, developers should use a class called CelloSaaS.View.Controls.CelloGrid to construct the grid. The structure of CelloSaaS.View.Controls.CelloGrid is as follows.

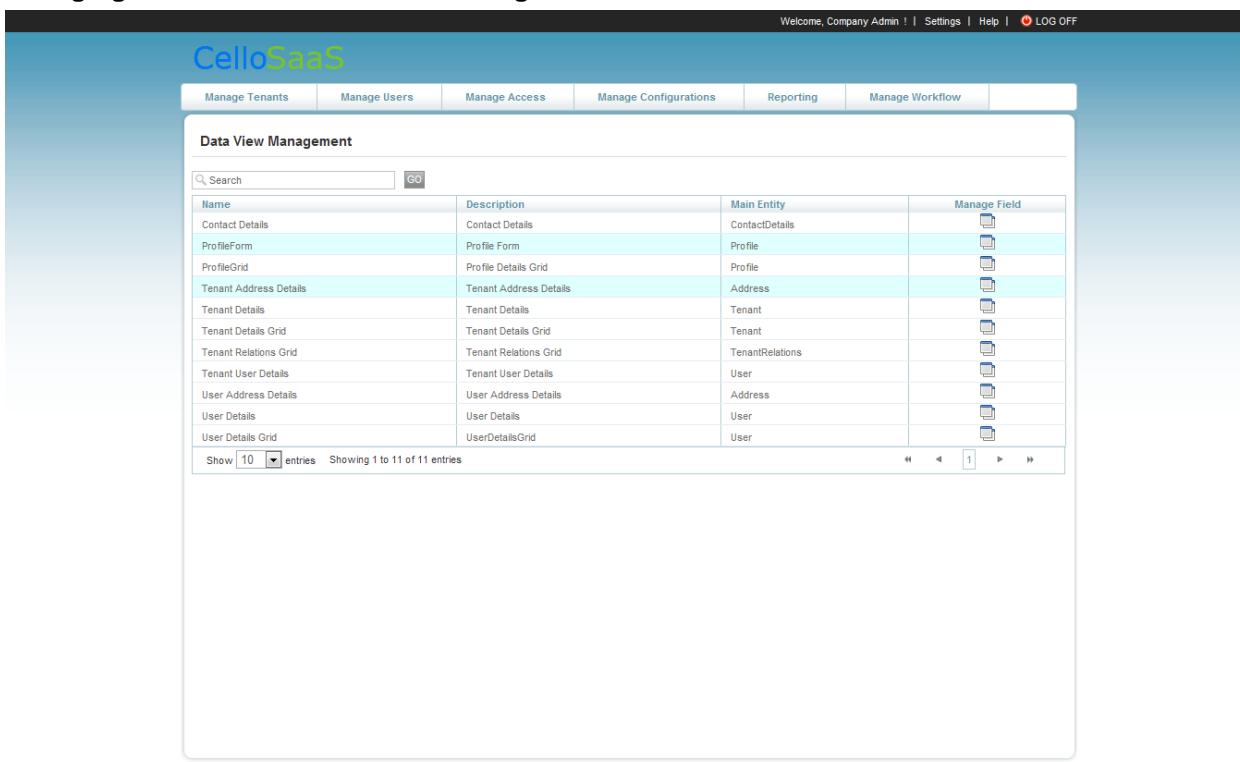
```
///<summary>
/// Creates a new instance of the Grid class.
///</summary>
///<param name="dataSource">The datasource for the grid</param>
///<param name="writer">TextWriter where the grid should be rendered</param>
///<param name="context">Html Viewcontext</param>
///<param name="dataViewID">Dataview Id</param>
CelloSaaS.View.Controls.CelloGrid.CelloGrid(IEnumerable<T> dataSource, TextWriter writer,
ViewContext context, string dataViewID)
///<summary>
/// This method is used to form the columns for grid by using the given column builder and it includes
the extented columns.
///</summary>
///<param name="celloColumnBuilder">Which has base columns</param>
///<returns></returns>
IGridWithOptions<T> CelloColumns(Action<celloColumnBuilder<T>> celloColumnBuilder)
```

Example

```
New CelloGrid<EmployeeDetails>((List< EmployeeDetails>) ViewData["EmployeeList"],
this.Html.ViewContext.HttpContext.Response.Output, this.Html.ViewContext,
"GirdManageEmployee").celloColumns(
column =>
{
    column.ForColumn(col =>
        col.EmployeeNumber).SetFieldIdentifier("EmployeeId").Named("Employee Id");
    column.ForColumn(col => col.FirstName).SetFieldIdentifier("First
Name").Named("First Name");
    column.ForColumn(col =>
        col.LastName).SetFieldIdentifier("LastName").Named("Last Name");
}).Render()
;
```

Here you need to pass dataviewId to the constructore of the cello Grid, DataViewId is the id created in DataviewTable. In each column you will have to set the FieldIdentifier. CelloSaaS will take care of the customization and extension of field implicitly.

Managing form/Grid Customization using User Interface



The screenshot shows the 'Data View Management' section of the CelloSaaS interface. At the top, there's a navigation bar with links for 'Manage Tenants', 'Manage Users', 'Manage Access', 'Manage Configurations', 'Reporting', and 'Manage Workflow'. To the right of the navigation is a 'Welcome, Company Admin ! | Settings | Help | LOG OFF' link.

The main area is titled 'Data View Management' and contains a search bar with a 'GO' button. Below the search bar is a table with four columns: 'Name', 'Description', 'Main Entity', and 'Manage Field'. The table lists various data views:

Name	Description	Main Entity	Manage Field
Contact Details	Contact Details	ContactDetails	
ProfileForm	Profile Form	Profile	
ProfileGrid	Profile Details Grid	Profile	
Tenant Address Details	Tenant Address Details	Address	
Tenant Details	Tenant Details	Tenant	
Tenant Details Grid	Tenant Details Grid	Tenant	
Tenant Relations Grid	Tenant Relations Grid	TenantRelations	
Tenant User Details	Tenant User Details	User	
User Address Details	User Address Details	Address	
User Details	User Details	User	
User Details Grid	UserDetailsGrid	User	

At the bottom of the table, there are pagination controls: 'Show 10 entries' and 'Showing 1 to 11 of 11 entries'.



Copyright © 2012 by techcello.com

Figure 6-15 – Data View Management Screen

The figure below displays the list of base fields and extended fields. You can include and exclude the extended fields and edit properties of both fields.

CelloSaaS

Manage Tenants | Manage Users | Manage Access | Manage Configurations | Reporting | Workflow | Phone Support | Re-Order | Back

UserAddressDetailsForm - Field Details

Display Name	Field Type	Ordinal	Is Visible	Is Extended Field	IsEditable	Edit Field	Exclude
Mobile Number	TextBox	0	False	True	False	Include	-
Fixed Line	TextBox	0	False	True	False	Include	-
Fax	TextBox	0	False	True	False	Include	-
Address	TextBox	1	True	False	-		-
City	TextBox	2	True	False	-		-
State	TextBox	3	True	False	-		-
Country	DropdownList	4	True	False	-		-
Postal Code	TextBox	5	True	False	-		-

Show 10 entries Showing 1 to 8 of 8 entries

Figure 6-16– UserAddressDetailsForm – Field Details Screen

The figure below displays the customization of the extended fields. Here you can set the properties like Display name, max length, regular expression, Field Type, Is Editable, Is Mandatory, Is visible, and Is Multiline, etc.



Welcome, Company Admin ! | Settings | Help | LOG OFF

CelloSaaS

Manage Tenants Manage Users Manage Access Manage Configurations Reporting Manage Workflow

DataView Field Properties

Display Name	CA_Test	IsExtendedField	True
MaxLength	0	IsEditable	<input checked="" type="checkbox"/>
RegularExpression		IsMandatory	<input type="checkbox"/>
Field Type	TextBox	IsVisible	<input checked="" type="checkbox"/>
Description			

Update Back



Copyright © 2012 by techcello.com

Figure 6-17– DataView Field Properties Screen

The figure below displays the customization of core fields. Here you can set the properties display name, visibility.

Welcome, Company Admin ! | Settings | Help |  LOG OFF

CelloSaaS

Manage Tenants | Manage Users | Manage Access | Manage Configurations | Reporting | Manage Workflow

Manage Field Details

Name *

PickUpField

Validation regular expression

Data type

Length

Is Unique

Back **Add**

Figure 6-18 – Manage Field Details Screen

CelloSaaS supports following Data Types,

Varchar

Int

Float

Boolean

Date

Pickup Field

Each Data type has the following properties.

Properties	Description	Supporting Data Types
Length	Specify the size of the data.	Varchar.
Regular Expression	Specify the regular expression for data.	Varchar, Int, Float and Date.
Unique	Specify the data is unique or not.	Varchar, Int, Float, Date and Pickup Field.

CelloSaaS supports following UI field types with corresponding data types.

Field Types	Data Types
-------------	------------

Textbox	varchar,Int,float
TextArea	Varchar
Calendar	Date
Dropdown	Pickuplist
Cascade Dropdownlist	cascade pickuplist
RadioButton	Boolean

CelloSaaS supports to set the regular expression to the varchar extended field. You can set the range (i.e. max and min value) to int, float and date field.

Syntax

Varchar

Character	Description	Example
Any character except [\^\$. ?*+()]	All characters except the listed special characters match a single instance of themselves. { and } are literal characters, unless they're part of a valid regular expression token (e.g. the {n} quantifier).	\a matches a
\ (backslash) followed by any of [\^\$. ?*+(){}]	A backslash escapes special characters to suppress their special meaning.	\+ matches +
\Q...\\E	Matches the characters between \Q and \E literally, suppressing the meaning of special characters.	\Q+-*/\E matches +-*/
\xFF where FF are 2 hexadecimal digits	Matches the character with the specified ASCII/ANSI value, which depends on the code page used. Can be used in character classes.	\xA9 matches © when using the Latin-1 code page.
\n, \r and \t	Match an LF character, CR character and a tab character respectively. Can be used in character classes.	\r\n matches a DOS/Windows CRLF line break.
\a, \e, \f and \v	Match a bell character (\x07), escape character (\x1B), form feed (\x0C) and vertical tab (\x0B) respectively. Can be used in character classes.	
\cA through \cZ	Match an ASCII character Control+A through Control+Z, equivalent to \x01 through \x1A. Can be used in character classes.	\cM\cJ matches a DOS/Windows CRLF line break.
[(opening square bracket)	Starts a character class. A character class matches a single character out of all the possibilities offered by the character class. Inside a character class,	

	different rules apply. The rules in this section are only valid inside character classes. The rules outside this section are not valid in character classes, except for a few character escapes that are indicated with "can be used inside character classes".	
Any character except ^-]\ add that character to the possible matches for the character class.	All characters except the listed special characters.	[abc] matches a, b or c
\ (backslash) followed by any of ^-]\	A backslash escapes special characters to suppress their special meaning.	[^\]] matches ^ or]
- (hyphen) except immediately after the opening [Specifies a range of characters. (Specifies a hyphen if placed immediately after the opening [)	[a-zA-Z0-9] matches any letter or digit
^ (caret) immediately after the opening [Negates the character class, causing it to match a single character notlisted in the character class. (Specifies a caret if placed anywhere except after the opening [)	[^a-d] matches x (any character except a, b, c or d)
\d, \w and \s	Shorthand character classes matching digits, word characters (letters, digits, and underscores), and whitespace (spaces, tabs, and line breaks). Can be used inside and outside character classes.	[\d\s] matches a character that is a digit or whitespace
\D, \W and \S	Negated versions of the above. Should be used only outside character classes. (Can be used inside, but that is confusing.)	\D matches a character that is not a digit
[\b]	Inside a character class, \b is a backspace character.	[\b\t] matches a backspace or tab character
Int,Float,Date		
>min value,<max value	Use angle brackets to specify the max and min value with coma	> 10, <100
<max value,>min value	You can give in this order also	> 100, <10

6.4 Pickup List Management

Pickup list represents a drop-down list attribute in an entity instance. SaaS providers need to provide their tenants the ability to manage their pickup lists and their values. CelloSaaS offers “Pickup List Management” feature to help the developers build this support easily to their product.

Pickup list can be used as an extended field in a form or a core field in an existing form.

Pickup list represents a drop-down list attribute in an entity instance.

SaaS providers need to provide their tenants the ability to manage their pickup lists and their values. CelloSaaS offers “Pickup List Management” feature to help the developers build this support easily to their product.

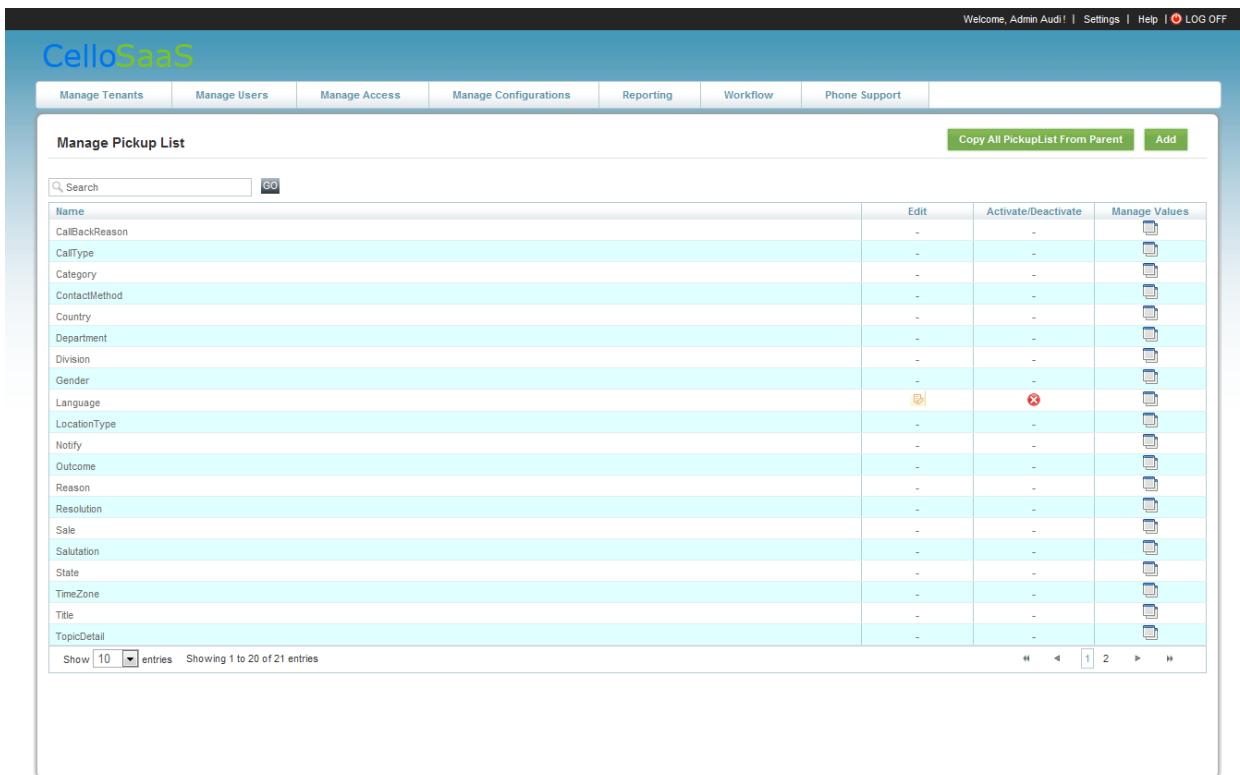
Pickup list can be used as an extended field in a form or a core field in an existing form.

Developer Usage

Generally pickup lists can be used to map with extended fields and fixed fields. If the pickup list is used for any of the development time core fields, following methods need to be used.

Pickup Lists

Pickup List details contain global and tenant based pickup lists.



Name	Edit	Activate/Deactivate	Manage Values
CallBackReason	-	-	
CallType	-	-	
Category	-	-	
ContactMethod	-	-	
Country	-	-	
Department	-	-	
Division	-	-	
Gender	-	-	
Language			
LocationType	-	-	
Notify	-	-	
Outcome	-	-	
Reason	-	-	
Resolution	-	-	
Sale	-	-	
Salutation	-	-	
State	-	-	
TimeZone	-	-	
Title	-	-	
TopicDetail	-	-	

Show 10 entries Showing 1 to 20 of 21 entries

Figure 6-19– Manage Pickup List Screen

The method below will return both tenant based and general pickup list.

```
///<summary>
/// This method is used to get the pick up list based on the tenant
///</summary>
///<param name="tenantCode">Tenant Code</param>
///<returns>Pick up list values</returns>
publicList<CelloSaaS.Model.Configuration.PickupList> GetPickupLists(string tenantCode)
```

```
To Get particular pickup list details
///<summary>
/// This method is used to get the pick up list details
///</summary>
///<param name="pickupListId">Pick Up list Id</param>
///<returns>Pickup list details</returns>
publicCelloSaaS.Model.Configuration.PickupList GetPickupListDetails(string pickupListId)
```

6.4.1 Managing Pickup Lists

Manage Configurations→ Configuration→ Manage Pickup Lists

Welcome Admin Audit | Settings | Help | LOG OFF

CelloSaaS

Manage Tenants | Manage Users | Manage Access | Manage Configurations | Reporting | Workflow | Phone Support

Manage Pickup List

Copy All PickupList From Parent | Add

Name	Edit	Activate/Deactivate	Manage Values
CallBackReason	-	-	
CallType	-	-	
Category	-	-	
ContactMethod	-	-	
Country	-	-	
Department	-	-	
Division	-	-	
Gender	-	-	
Language			
LocationType	-	-	
Notify	-	-	
Outcome	-	-	
Reason	-	-	
Resolution	-	-	
Sale	-	-	
Salutation	-	-	
State	-	-	
TimeZone	-	-	
Title	-	-	
TopicDetail	-	-	

Show 10 entries Showing 1 to 20 of 21 entries

Figure 6-20 – Manage Pickup List Screen

The values you are adding are the labels that will be displayed in your pickup list. You can change the values without affecting the stored value. After entering the values for the pickup list, specify whether the values are displayed in alphabetical order.

```
///<summary>
/// This method is used to insert Pickup list
///</summary>
///<param name="pickupList">Pickup list details</param>
/// Pickup list name mandatory
///<returns>Inserted Pickup list Id</returns>
public string AddPickupList(CelloSaaS.Model.Configuration.PickupList pickupList)
```



You can edit pickup lists when the field is in production. Be aware of the potential implications of this action. Make sure to provide a unique pickup list name. Avoid specifying a global pickup list name.

6.4.2 Editing Pickup Lists

If you choose to edit a pickup list value, the platform will update existing records with the new value.

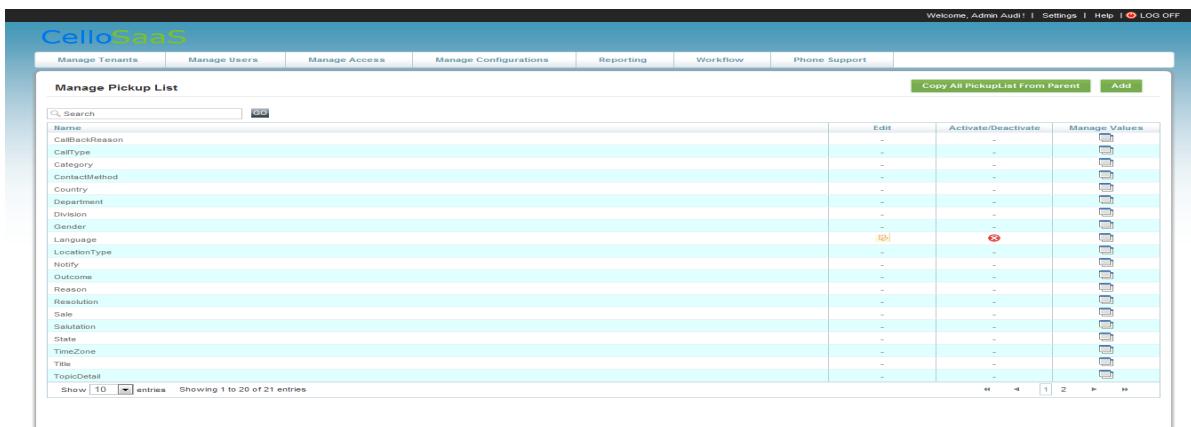


Figure 6-21 – Manage Pickup List Screen

```
To edit pickup list
///<summary>
/// This method is used to edit Pickup list
///</summary>
///<param name="pickupList">Pickup list details</param>
/// Pickup list Id and Pickup List name mandatory
///<returns>Pickup list Id</returns>
publicstring UpdatePickupList(CelloSaaS.Model.Configuration.PickupList pickupList)
```

6.4.3 Deactivate Pickup Lists

If you choose to deactivate a pickup list value, the application pops-up a confirmation message. Click Ok to proceed further.

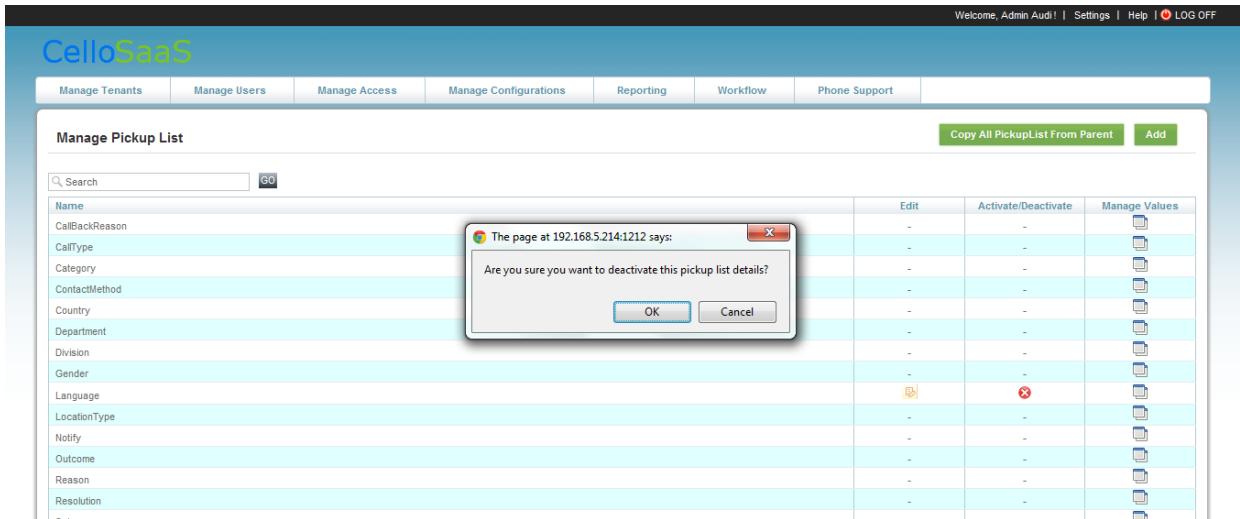


Figure 6-22 – Manage Pickup List Screen

```
///<summary>
/// This method is used to delete pick up list details
///</summary>
///<param name="pickupListId">Pickup list id</param>
publicvoid DeactivatePickupList(string pickupListId)
```

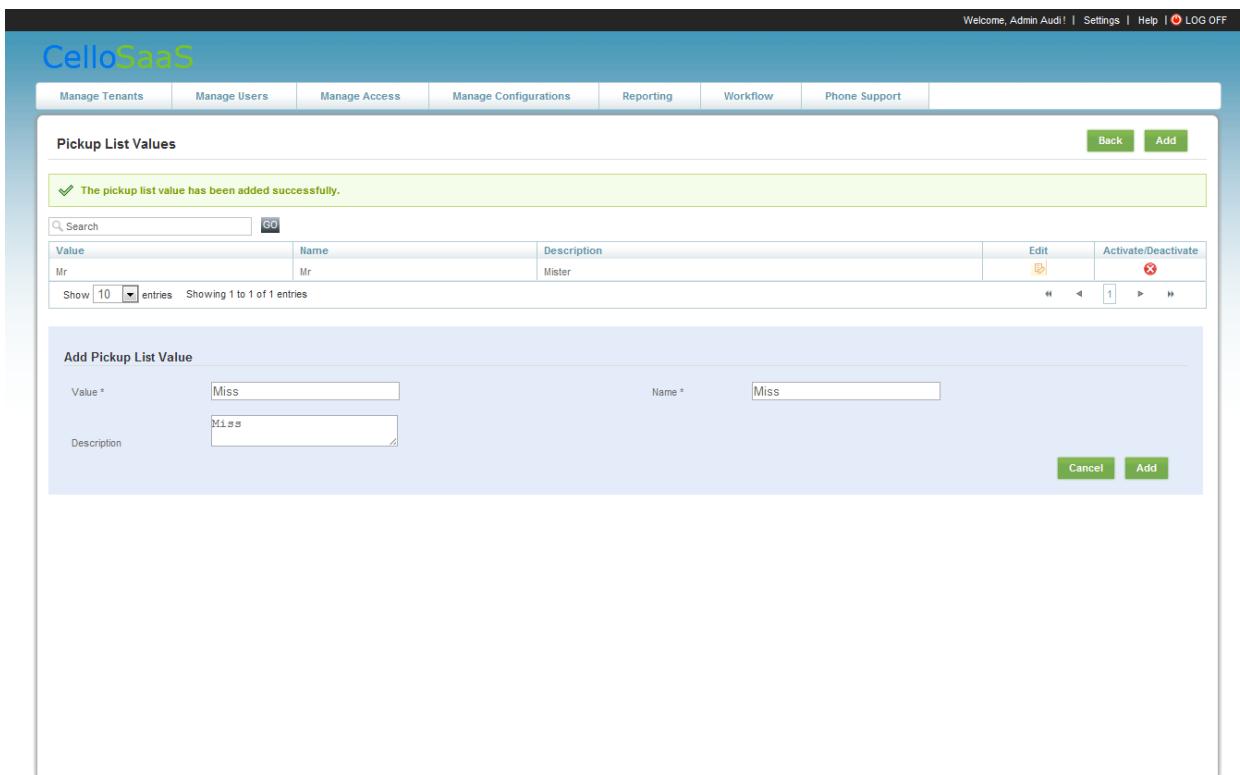
6.4.4 Pickup List value

Pickup List value has Item Id and Item Name. These two properties are used in the dropdown list like Item Id as Key and Item Name as Value.

General pickup list has some default values. To get the default values, select *copy from general pickup list* this will copy all default values into tenant based.



You will not be able to get the default values, if you have added a value in pickup list value before **copy from general pickup list**.



Value	Name	Description	Edit	Activate/Deactivate
Mr	Mr	Mister		

Show 10 entries Showing 1 to 1 of 1 entries

Add Pickup List Value

Value * Miss Name * Miss

Description Miss

Cancel Add

Figure 6-23 – Pickup List Values Screen

Pickup List value list

```
To get Pickup list value details based on the pickup list and tenant code
///<summary>
/// This method is used to get the pick up list based on the tenant
///</summary>
///<param name="tenantCode">Tenant Code</param>
///<returns>Pick up list values</returns>
publicList< CelloSaaS.Model.Configuration.PickupList> GetPickupLists(string tenantCode)
```

```
To Get particular pickup list value details
///<summary>
/// This method is used to get the pick up list details
///</summary>
```

```
///<param name="pickupListId">Pick Up list Identifier</param>
///<returns>Pickup list details</returns>
public CelloSaaS.Model.Configuration.PickupList GetPickupListDetails(string pickupListId)
```

Adding Pickup List Value

Create a pickup list with Name, appropriate description, and value and click Save button to add to the pickup list.

```
Add Pickup List Value
///<summary>
/// This method is used to insert Pickup list values
///</summary>
///<param name="pickupListValue">Pickup list value details</param>
/// Item Id, Item Value, Pickup List ID - Mandatory
///<returns>Pickup list value Id</returns>
public string AddPickUPLValue(CelloSaaS.Model.Configuration.PickupListValue pickupListValue)
```

Editing Pickup List Value

If you choose to edit a pickup list value, the platform will update existing records with the new value.

```
To Edit Pickup List
///<summary>
/// This method is used to edit Pickup list values
///</summary>
///<param name="pickupListValue">Pickup list value details</param>
/// Pickup List Value Id,Item Id,Item Value, Pickup List ID - Mandatory
///<returns>Pickup list value Id</returns>
public string UpdatePickupListValue(CelloSaaS.Model.Configuration.PickupListValue pickupListValue)
```

Deactivate Pickup List Value

If you choose to deactivate a pickup list value, the applications will pop-up for a confirmation message. Click Ok to proceed further.

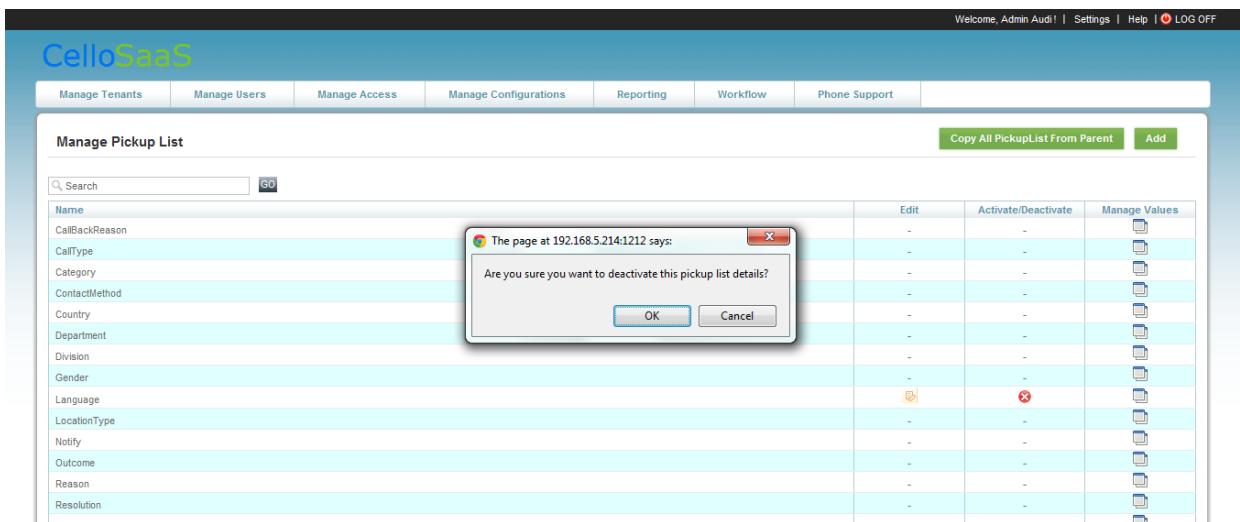


Figure 6-24 – Pickup List Values Screen

```
To Deactivate Pickup List Value
///<summary>
/// This method is used to deactivate pick up list value details
///</summary>
///<param name="pickupListValueId">Pick up list value id</param>
///<param name="pickupListId">Pick up list Id</param>
///<param name="tenantCode">Tenant Code</param>
Publicvoid DeactivatePickupListValue(string pickupListValueId, string pickupListId, string tenantCode)
```

Cascade Pickup List

The platform supports variation of parent-child relationship. Dependent picklists work in pairs, with the value selected in the parent picklist controlling the values shown in the child picklist. Cascade Pickup List means the relationship between two or more pickup lists. CelloSaaS contains Cascade Pickup List controller to get the pickup list values. You need to include the cascade pickup list in your master pages.

```
<script type="text/javascript" src="../../Scripts/CascadePickupList.js"></script>
```

A dependent picklist works in conjunction with a controlling picklist or checkbox to filter the available options. The value chosen in the controlling field affects the values available in the dependent field.

Relationship Details

Relationship details are maintained in the RelationshipMetaData table.

Relationship details contains list of PickupList/Entity relation based on the tenant.

CelloSaaS.Model.Configuration.RelationshipMetaData model having following details.

Relationship Details List

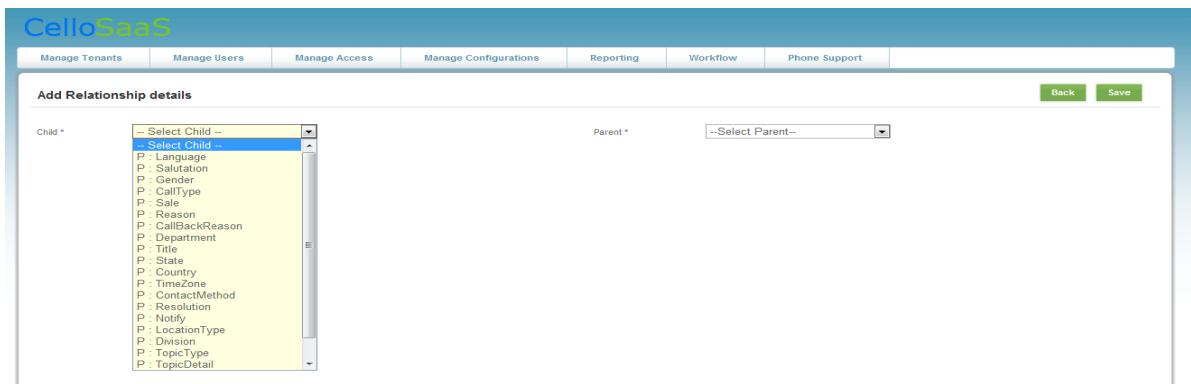
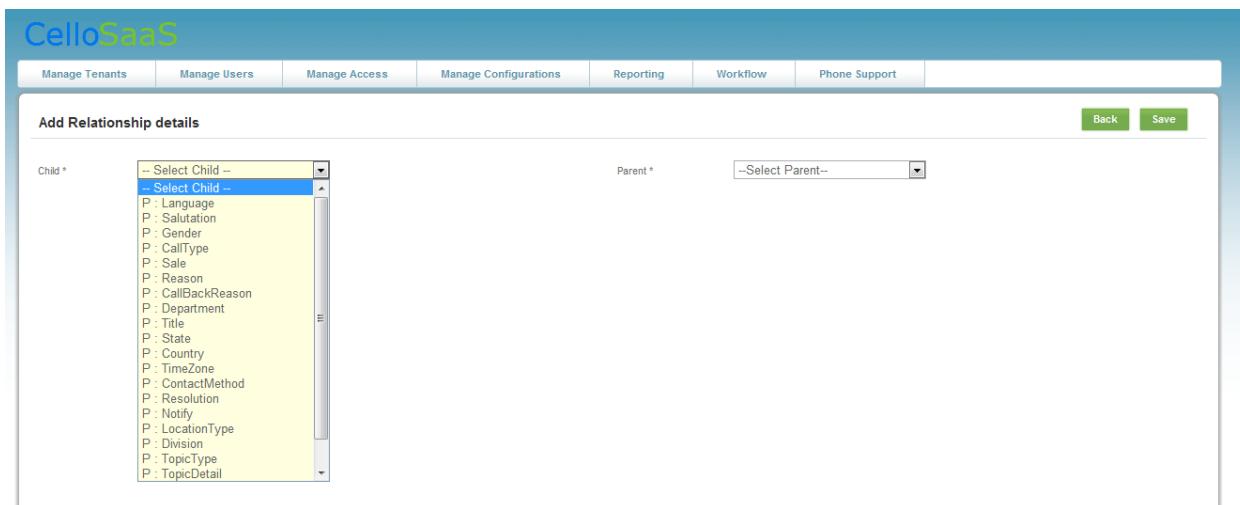


Figure 6-25 – Add Relationship Details Screen



Duplicate Relation / Hard Delete are not allowed.

To add Relationship Details



The screenshot shows the 'Add Relationship details' screen. At the top, there is a navigation bar with links: Manage Tenants, Manage Users, Manage Access, Manage Configurations, Reporting, Workflow, and Phone Support. Below the navigation bar, the main title is 'Add Relationship details'. On the left, there is a 'Child *' dropdown menu with a list of items. On the right, there is a 'Parent *' dropdown menu. At the top right of the form, there are 'Back' and 'Save' buttons.

Figure 6-26 – Add Relationship Details Screen

```

To Add Relationship
///<summary>
/// This method is used to add relationship meta data details
///</summary>
///<param name="relationshipMetaData">Relationship Meta Data Details</param>
/// Source Id, Target Id, Target Type, Relation type and Tenant Id - Mandatory
///<returns>Inserted Relationship Meta Data Id</returns>
public string AddRelationshipMetaData(CelloSaaS.Model.Configuration.RelationshipMetaData
relationshipMetaData)

```

Example

```

//Create instance for Picuplist service to access pickup list methods
CelloSaaS.ServiceContracts.Configuration.IPickupListService pickupListService =
(CelloSaaS.ServiceContracts.Configuration.IPickupListService)
CelloSaaS.Library.ServiceLocator.GetServiceImplementation(typeof(CelloSaaS.ServiceContracts.C
onfiguration.IPickupListService));
    //Call Get Relationship Meta Data method to get relationship details
string relationshipId = pickupListService.AddRelationshipMetaData(relationshipMetaData);

```

Example

Consider State and Country as two pickup Lists

Before insert relationship details



Welcome, Admin Audit! | Settings | Help | LOG OFF

CelloSaaS

- Manage Tenants
- Manage Users
- Manage Access
- Manage Configurations
- Reporting
- Workflow
- Phone Support

Pickup List Values

The pickup list value has been added successfully.

Value	Name	Description	Edit	Activate/Deactivate
Mr	Mr	Mister		

Show 10 entries Showing 1 to 1 of 1 entries

Add Pickup List Value

Value *	Miss	Name *	Miss
Description	Miss		

Cancel **Add**

Copyright © 2012 by techcello.com

Figure 6-27 – Pickup List Values Screen

Welcome, Admin Audit! | Settings | Help | LOG OFF

CelloSaaS

- Manage Tenants
- Manage Users
- Manage Access
- Manage Configurations
- Reporting
- Workflow
- Phone Support

Pickup List Values

The pickup list value has been added successfully.

Value	Name	Description	Edit	Activate/Deactivate
Mr	Mr	Mister		

Show 10 entries Showing 1 to 1 of 1 entries

Add Pickup List Value

Value *	Miss	Name *	Miss
Description	Miss		

Cancel **Add**

Copyright © 2012 by techcello.com

Figure 6-28 – Add Pickup List Values Screen

This relationship values will be stored in the PickupListValueMapping table.

Adding Cascading Pickup List as Extended Field

While Adding Extended Fields in the Forms Customization you will get two field types, dropdown list and cascade dropdown list.

When you select cascade dropdown list, it will check if any relationship details are available for the pickup field.

If there is no relationship available, you cannot select cascade dropdown list type and you will get alert message.

If relationship details are available then the parent pickup list details will be displayed in the Parent field.

Example

Assume we are extending the contact details form.

Include state as dropdown list in Contact details form.

Include City as Cascade dropdown list and select state as parent field.

Now Contact details form will contain State and City.

When you change the state value then the corresponding city value will be rendered.

Contact Details			
First Name	<input type="text"/>	Last Name	<input type="text"/>
Phone	<input type="text"/>	Fax	<input type="text"/>
Email	<input type="text"/>	Mobile Number	<input type="text"/>
State	<input type="text" value="Delhi"/>	City	<input type="text" value="-- Select --"/> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;"> ...Select... -Select- Chennai Trichy Kovai Devakottai </div>
Company User Details			
First Name	<input type="text"/>	Last Name	<input type="text"/>
User Name	<input type="text"/>		

Figure 6-29 – Contact Details / Company User Details Screen

CelloSaaS Service Endpoint Management

7 CELLOSAAS SERVICE ENDPOINT MANAGEMENT

CelloSaaS service endpoint management feature allows the application to dynamically register Web services and use them. Typical scenario is when your application need to confirm or send some data to third party application where the third party service endpoint address is not known at design time or configurable per tenant.

APIs

```
interface CelloSaaS.Integration.ServiceContracts.ICelloServiceEndpointManager

/// <summary>
/// Adds the service endpoint metadata to database.
/// </summary>
/// <param name="endpoint">The endpoint.</param>
/// <returns></returns>
string AddServiceEndpoint(CelloServiceEndpoint endpoint);

/// <summary>
/// Update the service endpoint metadata in the database.
/// </summary>
/// <param name="endpoint">Endpoint metadata to update.</param>
void UpdateServiceEndpoint(CelloServiceEndpoint endpoint);

/// <summary>
/// Deletes the given endpoint metadata identifier from the database
/// </summary>
/// <param name="endpointId">Endpoint identifier</param>
/// <param name="tenantId">Tenant identifier</param>
void DeleteServiceEndpoint(string endpointId, string tenantId);

/// <summary>
/// Gets the service endpoint metadata from the database
/// </summary>
/// <param name="endpointId"></param>
/// <param name="tenantId"></param>
/// <returns></returns>
CelloServiceEndpoint GetServiceEndpoint(string endpointId, string tenantId);

/// <summary>
/// Searches the endpoint metadata by given endpoint name in the database.
/// </summary>
/// <param name="name">endpoint name to search.</param>
/// <param name="tenantId">tenant identifier</param>
/// <returns></returns>
CelloServiceEndpoint GetServiceEndpointByName(string name, string tenantId);

/// <summary>
/// Searches all the endpoint metadata's for the given tenant identifier
/// </summary>
/// <param name="tenantId">tenant identifier</param>
/// <returns></returns>
Dictionary<string, CelloServiceEndpoint> GetServiceEndpointByTenantId(string tenantId);
```



```
/// <summary>
/// Calls the given web service with the inputXml and returns the output xml.
/// </summary>
/// <param name="serviceEndpointId">service endpoint primary key</param>
/// <param name="tenantId">tenant identifier</param>
/// <param name="inputXml">body xml string</param>
/// <returns></returns>
string InvokeServiceById(string serviceEndpointId, string tenantId, string inputXml);

/// <summary>
/// Calls the given web service with the inputXml and returns the output xml.
/// </summary>
/// <param name="serviceEndpointName">service endpoint name</param>
/// <param name="tenantId">tenant identifier</param>
/// <param name="inputXml">body xml string</param>
/// <returns></returns>
string InvokeServiceByName(string serviceEndpointName, string tenantId, string inputXml);

/// <summary>
/// Invokes web service directly from given endpoint url and xml.
/// </summary>
/// <param name="endpointDetails">service endpoint url</param>
/// <param name="inputXml">input xml.</param>
/// <returns></returns>
string InvokeService(CelloServiceEndpoint endpointDetails, string inputXml);
```

Example Usage

```
var serviceMetadataManager = ServiceLocator.Resolve<ICelloServiceEndpointManager>();
var outputXml = serviceMetadataManager.InvokeServiceById(serviceEndpointId, tenantId,
inputXml);
```

Supported Security

CelloSaaS support the below mentioned security when calling the SOAP based web service

UserName & password

The remote service is authenticated via the supplied username and password or logged in CelloSaaS username and password.

CelloSaaS Shared WCF Key

The remote service is authenticated by the cellosaas logged in username/tenant and shared secret key specific to tenant.

Security Token (Active Federated Authentication)

The remote service is authenticated by the security token which is acuired from the configured ADFS/Custom STS service prior to calling the remote service. The ADFS/Custom STS is authenticated by username & password authentication.

Both http & https endpoints are supported. Encryption certificate should be exported as .CER base64 encoded string and specified in the "ServiceCertificate" property.

For REST based services HTTP basic security is not supported in this version (4.2). The service must take care of authentication via POST or Query String parameters.



Sample

Create endpoint details

Endpoint name should be unique(tenant wise). Below are mandatory/necessary properties defined to define a SOAP service endpoint.

```
var endpointDetails = new CelloServiceEndpoint
{
    Name = "Get Product details",
    Description = "Get product details using username authentication and message security.",
    EndpointAddress = "http://remote.endpointaddress/ProductService.svc",
    SoapMetadata = new SoapMetadata
    {
        SoapAction = "http://tempuri.org/IProductService/GetProductById",
        CredentialType = CredentialType.UserName,
        ServiceCertificate = @"MIIC8jCCAdqgAwIBAgIQH/eu3wHpE4hFSfP110/....",
    },
    TenantId = UserIdentity.TenantID,
    CreatedBy = UserIdentity.UserId,
    Status=true
};

string id = CelloServiceEndpointProxy.AddServiceEndpoint(endpointDetails);
```

Note: When calling this web service logged in CelloSaaS UserName and password is used to authenticate. ServiceCertificate is base64 encoded certificate key string for brevity complete string is not given. If you want to override the CelloSaaS UserName then set the UserName and Password property as below.

```
endpointDetails.SoapMetadata.UserName = "scott";
endpointDetails.SoapMetadata.Password = "tigger";
```

Call the service

```
var inputXml = @"<GetProductById xmlns=""http://tempuri.org/"">
    <id>4C10CE01-6405-E211-8AE1-000000000000</id>
</GetProductById>";
var outputXml = CelloServiceEndpointProxy.InvokeServiceByName("Get Product details",
UserIdentity.TenantID, inputXml);
```

Note: inputXml should be a valid SOAP body xml. First element should be ActionName (method name) and should contain valid namespace to work correctly.

The outputXml will be the response SOAP body xml.

Sample Output Xml:

```
<GetProductByIdResponse
    xmlns="http://tempuri.org/">
    <GetProductByIdResult
        xmlns:b="http://schemas.datacontract.org/2004/07/Ruf.Model"
        xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <TenantId
            xmlns="http://schemas.datacontract.org/2004/07/CelloSaaS.Model">b590cd25-
3093-df11-8deb-001ec9dab123
```

```

        </TenantId>
        <b:Barcode>1234567823</b:Barcode>
        <b:CreatedBy>Admin</b:CreatedBy>
        <b:CreatedOn>2012-05-05T00:00:00</b:CreatedOn>
        <b:Description>test</b:Description>
        <b:Discount>0.5</b:Discount>
        <b:Id>4c10ce01-6405-e211-8ael-000000000000</b:Id>
        <b:MRP>1000</b:MRP>
        <b:Name>CelloSaaS SaaS Framework</b:Name>
        <b:Price>999.989990234375</b:Price>
        <b>Status>true</b>Status>
        <b>TenantId>b590cd25-3093-df11-8deb-001ec9dab123</b:TenantId>
        <b:Unit>1 units</b:Unit>
        <b:UpdatedBy>3398f837-b988-4708-999d-d3dfe11875b3</b:UpdatedBy>
        <b:UpdatedOn>2012-09-24T20:04:55.207</b:UpdatedOn>
    </GetProductByIdResult>
</GetProductByIdResponse>

```

5. UI Screens

a) Service endpoint details list

Service Endpoint Management							
<input type="text" value="Search"/> <input type="button" value="GO"/>						<input type="button" value="Add"/>	
Name	Description	Endpoint Address	Credential Type	Test	Edit	Delete	
Get Product details	get product details via saml token and custom username & password	https://[REDACTED]:4444/ProductService.svc	UserName	Test			
Get Product via ADFS	get product details via adfs security token.	https://[REDACTED]:4444/ProductService.svc	CelloSharedKey	Test			
GetProductById	get product entity via custom STS.	https://[REDACTED]:4444/ProductService.svc	CelloSharedKey	Test			
GetUserDetails	get user details by user id via default username, shared key.	https://localhost:4444/Usermanagement/UserDetailsService.svc	CelloSharedKey	Test			

Show 10 entries Showing 1 to 4 of 4 entries

b) Add/Edit Endpoint details

Edit Service Endpoint

Save **Back**

Name*	<input type="text" value="GetProductById"/>	Description	<input type="text" value="get product entity via custom STS."/>
Endpoint URL*	<input type="text" value="https://...com:44"/>	SOAP Action*	<input type="text" value="http://tempuri.org/IProductService/GetProduct"/>
Message Modifier Address	<input type="text" value="https://...com:44"/>	Client Credential Type	<input type="text" value="CelloSharedKey"/>
Encryption Certificate (Base64 Encoded Key)	<pre>MII08jCCAdqgAwIBAgIQH/eu3wHpE4hFSIP110/ZODANBgkqhkiG9w0BAQUFADAiMSAwHgYDVQQDEdhc3BpcmU0NzUuQXNwaXJlc3IzMnVbTAeFw0xMjA5MjYxNDMxMzVaFw0xMzA5MjYwMDAwMDBaMCIxIDAeBgNVBAMTF2FzcGlyZTQ3NS5Bc3BpcmVzeXMuY29tMIIIBjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBCgKCAQEAxCR83UklKhN</pre>		
STS Endpoint or Federation Endpoint URL	<input type="text" value="https://...com:44"/>	Federation Realm Endpoint URL	<input type="text"/>
Security Key Type	<input type="text" value="SymmetricKey"/>	Negotiate Service Credential	<input type="checkbox"/>
Establish Security Context	<input type="checkbox"/>	Output Xml Schema	<input type="checkbox"/>
Input Xml Schema	<input type="text"/>		

Save **Back**

c) Testing Endpoint



Test Service

[Invoke](#) [Back](#)

Endpoint Details:

Name: GetProductById
Description: get product entity via custom STS.
Endpoint Address: https://[REDACTED].com:4444/ProductService.svc
SOAP Action: http://tempuri.org/IProductService/GetProductById

Input Xml:

```
<GetProductById xmlns="http://tempuri.org/">
<id>4C10CE01-6405-E211-8AE1-000000000000</id>
</GetProductById>
```

Output Xml:

```
<Identifier
  xmlns="http://schemas.datacontract.org/2004/07/CelloSaaS.Model">4c10ce01-6405-e211-8ae1-000000000000
<Identifier>
<IsTrackingOn
  xmlns="http://schemas.datacontract.org/2004/07/CelloSaaS.Model">false
</IsTrackingOn>
<GuidIdentifier
  xmlns="http://schemas.datacontract.org/2004/07/CelloSaaS.Model">00000000-0000-0000-0000-000000000000
</GuidIdentifier>
<IntegerIdentifier
  xmlns="http://schemas.datacontract.org/2004/07/CelloSaaS.Model">0
</IntegerIdentifier>
<TenantId
  xmlns="http://schemas.datacontract.org/2004/07/CelloSaaS.Model">b590cd25-3093-df11-8deb-001ec9dab123
</TenantId>
<b:Barcode>1234567823</b:Barcode>
<b:CreatedBy>Admin</b:CreatedBy>
<b:CreatedOn>2012-05-05T00:00:00</b:CreatedOn>
<b>Description>text</b>Description>
<b:Discount>5</b:Discount>
<b:Id>4c10ce01-6405-e211-8ae1-000000000000</b:Id>
<b:MRP>1000</b:MRP>
<b>Name>CelloSaaS SaaS Framework</b>Name>
<b>Price>999.999990234375</b>Price>
<b>Status>true</b>Status>
<b>TenantId>b590cd25-3093-df11-8deb-001ec9dab123</b>TenantId>
<b>Unit>1 units</b>Unit>
<b>UpdatedBy>3398f637-b988-4708-999d-d3dfe11875b3</b>UpdatedBy>
<b>UpdatedOn>2012-09-24T20:04:55.207</b>UpdatedOn>
</GetProductByIdResult>
</GetProductByIdResponse>
```

[Invoke](#) [Back](#)

8 SECURITY FRAMEWORK

Following are some of the security mechanisms followed in SaaS product. They are

Infrastructure security

Data security at database level

Data security at Field level

Data Security at Tenant Level

Data Security at Role Level

Data Security at record Level

Security at infrastructure level is generally handled at the infrastructure / environment setup level. Typically, infrastructure specialists will focus on VAPT to make sure the environment is secured and no unauthorized people will break into the system.

Data security at database level:

Largely focuses on making sure that a tenant has access only to their respective data and not others. This is handled in different ways for different database sharding (shared database, isolated database, etc.). Even in shared database model, framework guarantees that data security prevails at tenant level. CelloSaaS takes care of this requirement through the security framework.

Data security at Field level: Any SaaS product should offer the capability for their tenants to manage their access control policy at granular levels. CelloSaaS's "Security Framework" allows the developer to build this support in their product more easily. CelloSaaS supports configuring access for the following:

Pages

Entity Actions/ Generic Actions

Data

Fields

CelloSaaS achieves the access control using Privileges. Developers need to demand the right permissions in the code for Pages, Actions, Data and Fields and based on the tenant's privileges, access will be granted/not granted.

Privileges are given for Roles and Licenses. A privilege possessed by a user is an intersection of RolePrivilege and his/her tenant's License privileges.

Main Menu	First Level Menu	Second Level Menu	Privilege
Home			No Privilege
My Actions			No Privilege
	Membership Requests		No Privilege

Main Menu	First Level Menu	Second Level Menu	Privilege
	System Notifications		No Privilege
	Change Password		No Privilege
	Pending Tasks		No Privilege
Admin			P:View_Tenant OR P:View_Package OR P:View_TenantTemplate OR P:View_User OR S:ShareUsers OR P:View_Role OR P:Manage_TenantAccess OR P:Manage_Entity OR P:Manage_DataView OR P:View_SettingsTemplate OR P:Manage_ModuleConfiguration OR P:View_Rules OR P:View_PickupList OR R:GR\$Product_Admin OR P:View_Notification OR P:View_Audit OR P:View_Databackup OR R:GR\$Tenant_Admin
	Tenant Management		P:View_Tenant OR P:View_Package OR P:View_TenantTemplate
		Manage Tenant Details	P:View_Tenant
		Manage Package Details	P:View_Package
		Approve Child Tenants	P:View_Tenant
		Manage Tenant Templates	P:View_TenantTemplate
	User Management		P:View_User OR S:ShareUsers
		Manage UserDetails	P:View_User
		Manage External Users	S:ShareUsers
	Access Management		P:View_Role OR P:Manage_TenantAccess

Main Menu	First Level Menu	Second Level Menu	Privilege
		Manage Roles	P:View_Role
		Manage Tenant Access	P:Manage_TenantAccess
	Configuration Management		P:Manage_Entity OR P:Manage_DataView OR P:View_SettingsTemplate OR P:Manage_ModuleConfiguration OR P:View_Rules OR P:View_PickupList OR R:GR\$Product_Admin
		Manage Entities	P:Manage_Entity
		Manage Data View	P:Manage_DataView
		Manage Settings Templates	P:View_SettingsTemplate
		Module Configuration	P:Manage_ModuleConfiguration
		BusinessRules	P:View_Rules
		Manage Pickup List	P:View_PickupList
		Manage Relationship	P:View_PickupList
		Manage MasterData	R:GR\$Product_Admin
	Notification Management		P:View_Notification
		Notification Management	P:View_Notification
		Content Template	No Privilege
	Audits		P:View_Audit
		Product Analytics	P:View_Audit
		Notification	P:View_Audit AND P:View_Notification
		Event Audits	P:View_Audit

Main Menu	First Level Menu	Second Level Menu	Privilege
	Data Backup		P:View_Databackup
	Refresh cache		R:GR\$Product_Admin OR R:GR\$Tenant_Admin
Reporting			P:View_Query OR P:View_Chart OR P:View_Report
	Query Builder		P:View_Query
	Chart Builder		P:View_Chart
	Report Builder		P:View_Report
		Report Builder	P:View_Report
		Table Sources Builder	P:View_Report
Workflow			P:View_WorkflowDesign
	Workflow		P:View_WorkflowDesign
	WF Dashboard		P:View_WorkflowDesign
	Search Workflow		P:View_WorkflowDesign
	Search WFTaskInstance		P:View_WorkflowDesign
Manage Events			P:View_Audit
	Manage Events		P:View_Audit
	Content Template		No Privilege

8.1 Role Management

Every user can be mapped to one or many Roles. Role Management is the application area in CelloSaaS from where roles are easily set up and managed. Tenant can create roles and manage the privileges for their roles. Roles are flexible and can be customized to the exact requirements of any tenant.

Roles refer to the logical role the user play in the context of the application.

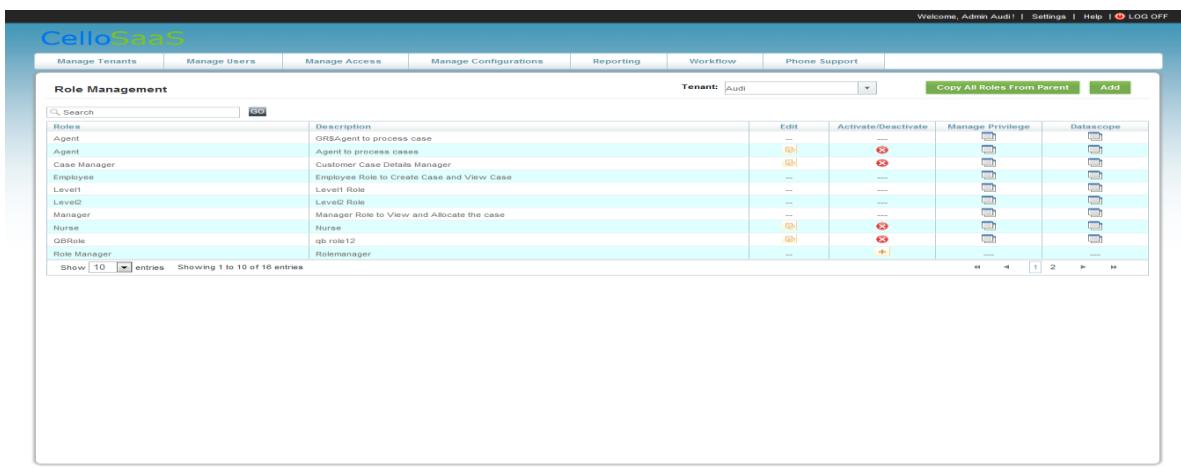
Each user can be assigned one or more roles.

At runtime, information about the logged in user is available in Context User Object.

8.1.1 Roles and Access provisions in CelloSaaS

Roles	Access provisions
Global Roles	<p>These roles are common to all tenants</p> <p>As of now CelloSaaS considers Tenant Admin as global role</p>
Tenant Based Roles	<p>Tenant Admin does not possess rights to edit/delete global roles</p> <p>Tenant based roles are created by Tenant Admin</p>
	<p>Specific to a particular tenant</p> <p>Product admin role is assigned to Company Admin</p>
	<p>Product admin role has permission to Add Tenant details</p>

Role Details List



The screenshot shows the 'Role Management' section of the CelloSaaS interface. The top navigation bar includes links for Manage Tenants, Manage Users, Manage Access, Manage Configurations, Reporting, Workflow, and Phone Support. A dropdown menu for 'Tenant' is set to 'Audi'. On the right, there are buttons for 'Copy All Roles From Parent' and 'Add'. The main area displays a table of roles:

Role	Description	Edit	Activate/Deactivate	Manage Privilege	Datascope
GRSAgent	GRSAgent to process case	—	—	—	—
Agent	Agent to process cases	—	—	—	—
Case Manager	Customer Case Details Manager	—	—	—	—
Employee	Employee Role to Create Case and View Case	—	—	—	—
Level0	Level0 Role	—	—	—	—
Manager	Manager Role to View and Allocate the case	—	—	—	—
Nurse	Nurse	—	—	—	—
GRBRole	qb role12	—	—	—	—
Role Manager	Rolemanager	—	—	—	—

At the bottom left, there are buttons for 'Show' and 'entries', with 'Showing 1 to 10 of 16 entries'. The footer of the page includes the Techcello logo and the copyright notice 'Copyright © 2012 by techcello.com'.

Figure 8-1 – Role Management Screen

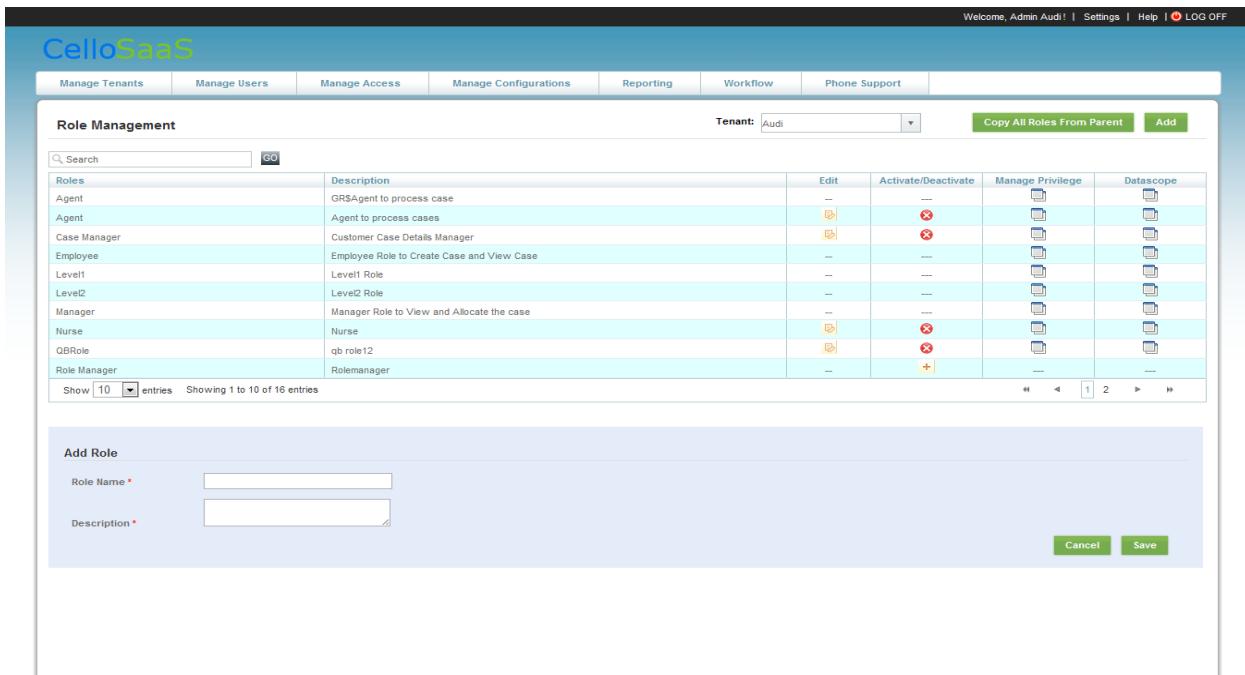
8.1.2 Add/Edit Role

By default, Tenant Admin have privileges to add role details

Role Name and Description are mandatory inputs

Role Name should be unique within tenant and should not be a Global Role Name

Role Name should not contain special characters except _ and -
 Role Id is auto generated with the format of TenantCode_RoleName



Roles	Description	Edit	Activate/Deactivate	Manage Privilege	Datascope
Agent	GRC Agent to process case	--	--		
Agent	Agent to process cases				
Case Manager	Customer Case Details Manager				
Employee	Employee Role to Create Case and View Case	--	--		
Level1	Level1 Role	--	--		
Level2	Level2 Role	--	--		
Manager	Manager Role to View and Allocate the case	--	--		
Nurse	Nurse				
QBRole	qb role12				
Role Manager	Rolemanager	--		--	--



Copyright © 2012 by techcello.com

Figure 8-2 – Role Management Screen

```
To Add role details
///<summary>
/// This method is used to add role
///</summary>
///<param name="tenantCode">Tenant Code</param>
///<param name="roleName">Role Name(Mandatory)</param>
///<param name="description">Role Description</param>
///<returns>Inserted Role Id</returns>
publicstring AddRole(string tenantCode, string roleName, string description)
///<summary>
/// This method is used to add role
///</summary>
///<param name="roleDetails">Role Details(Mandatory - Role Name)</param>
///<returns>Inserted Role Id</returns>
publicstring AddRole(Role roleDetails)
///<summary>
/// This method is used to add role with some condition
///</summary>
///<param name="tenantCode">Tenant Code</param>
///<param name="roleName">Role Name(Mandatory)</param>
///<param name="description">Role Description</param>
///<param name="roleQualifierQuery">Qualifier Query</param>
///<returns>Inserted Role Id</returns>
publicstring AddRole(string tenantCode, string roleName, string description, string
roleQualifierQuery)
///<summary>
/// This method is used to add role with some condition
```

```

///</summary>
///<param name="roleDetails">Role Details(Mandatory - Role Name)</param>
///<param name="roleQualifierQuery">Qualifier Query</param>
///<returns>Inserted Role Id</returns>
publicstring AddRole(Role roleDetails, string roleQualifierQuery)

```

To update role details

```

///<summary>
/// This method is used to update role details
///</summary>
///<param name="roleIdentifier">Role ID(Mandatory)</param>
///<param name="description">Role Description</param>
///<param name="qualifierQuery">Qualifier query</param>
///<returns>Updated Role ID</returns>
publicstring UpdateRole(string roleIdentifier, string description, string qualifierQuery)
///<summary>
/// This method is used to update role details
///</summary>
///<param name="roleDetails">Role Details(Mandatory - Role ID)</param>
///<param name="qualifierQuery">Qualifier query</param>
///<returns>Updated Role ID</returns>
publicstring UpdateRole(Role roleDetails, string qualifierQuery)

```

8.1.3 Delete Role

Admin can't delete global roles. They can delete only tenant based roles.

We have soft delete and permanent delete methods.

Soft Delete – Deactivate Role.

Permanent Delete – Delete Role. Here we are deleting all reference table data also.

Once a Role is created, any user can be assigned to that role.

To Hard Delete

```

///<summary>
/// This method is used to delete specified role details.
///</summary>
///<param name="tenantCode">Tenant Code</param>
///<param name="roleIdentifier">Role Identifier(Mandatory)</param>
publicvoid PermanentDeleteRole(string tenantCode, string roleIdentifier)

```

To Deactivate Role

```

///<summary>
/// This method is used to deactivate specified role details.
///</summary>
///<param name="tenantCode">Tenant Code</param>
///<param name="roleIdentifier">Role Identifier(Mandatory)</param>
publicvoid DeleteRole(string tenantCode, string roleIdentifier)

```

8.2 Service Administrator

Service Admin

Service admin is like Product Admin. Service administrator is usually assigned to administer a single or multiple services. This would be helpful in SaaS applications where there are multiple services (product lines). Service admin role will be added by default in the db script. This role will be used to manage users and roles based on service.



Consider an example of HR Management system which will have many services like Identity Management System, Leave Management System and Growth Management System. Each of these services will have separate administrator who will manage respective services.

Product administrator

Product administrator is a user who has the role “Product Admin”. This user will be created by default in the db script. This role is given the following privileges by default:

Creating service administrator

Creating Global role with/without service mapping

Assigning Privileges to roles.

Creating service administrator

Login as Product Admin and create a user in User Management

In Add Roles to user, you will have a role “**Service Admin**”. Once you click on service admin, list of service will be displayed after which you can map the services to User.

The screenshot shows the 'Add User Details' page of the CelloSaaS application. At the top, there's a navigation bar with links for Manage Tenants, Manage Users, Manage Access, Manage Configurations, Reporting, and Workflow. On the far right of the header, it says 'Welcome, Company Admin | Settings | Help | LOG OFF'. Below the header, the main form is titled 'Add User Details'. It has two main sections: 'User Details' and 'Address Details'. In the 'User Details' section, there are fields for First Name, Last Name, User Name, and Email. In the 'Address Details' section, there are fields for Address, City, State, Country, Postal Code, Language, and Contact Number. At the bottom right of the form, there are 'Cancel' and 'Save' buttons. The footer of the page includes the 'techcello' logo and the text 'Copyright © 2012 by techcello.com'.

Figure 8-3 – Add User Details Screen

You can also consume the following services for User-Service mapping which will be available in CelloSaaS.Services at CelloSaaS.Services.LicenseManagement.[LicenseService](#)

```

///<summary>
/// This method is used to get service details for given userId.
///</summary>
///<param name="userId"></param>
///<returns></returns>
Dictionary<string, Service> GetServicesByUserId(string userId);
///<summary>
/// Gets the service details for given userIds.
///</summary>
///<param name="userIds">array of user identifiers</param>
///<returns>dictionary of services mapped to the user id</returns>
Dictionary<string, Service> GetServicesByUserIds(string[] userIds);
///<summary>
/// This method is used to update user service.
///</summary>
///<param name="userId">the user identifier</param>
///<param name="serviceIds">array of service identifiers</param>
void UpdateUserServices(string userId, string[] serviceIds);
///<summary>
/// This method is used to delete user services.
///</summary>
///<param name="userId">the user identifier</param>
///<param name="serviceIds">array of service identifiers</param>
void DeleteUserServices(string userId, string[] serviceIds)

```

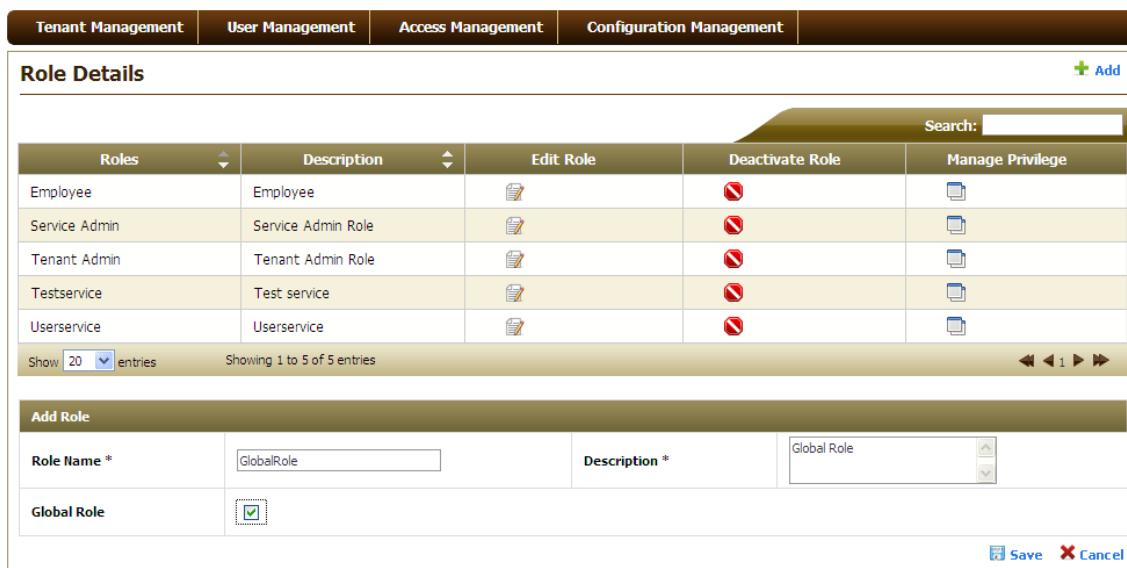
Creating Global role with/without service mapping

In Role details page, for product admin, all the global roles with and without service mapping, tenant based roles will be displayed.

The user can create global role with and without service mapping. The user can create tenant based role.

Once you check the Global Role checkbox, services which are mapped to logged-in Service Admin will be displayed.

Without service mapping



The screenshot shows the 'Role Details' section of the CelloSaaS interface. At the top, there are tabs for Tenant Management, User Management, Access Management, and Configuration Management. Below the tabs, a search bar and an 'Add' button are visible. The main area displays a table of roles with columns for Roles, Description, Edit Role, Deactivate Role, and Manage Privilege. The roles listed are Employee, Service Admin, Tenant Admin, Testservice, and Userservice. Each row has edit and deactivate icons. At the bottom of the table, there are pagination controls showing 20 entries and 1 to 5 of 5 total. Below the table, an 'Add Role' form is shown. It includes fields for Role Name (GlobalRole), Description (Global Role), and a Global Role checkbox which is checked. There are also 'Save' and 'Cancel' buttons at the bottom right of the form.

Roles	Description	Edit Role	Deactivate Role	Manage Privilege
Employee	Employee			
Service Admin	Service Admin Role			
Tenant Admin	Tenant Admin Role			
Testservice	Test service			
Userservice	Userservice			

Show 20 entries Showing 1 to 5 of 5 entries

Add Role

Role Name *: GlobalRole

Description *: Global Role

Global Role:

Save Cancel



Figure 8-4 – Role Details Screen

With service mapping

The screenshot shows the 'Manage User Roles' screen in the CelloSaaS application. At the top, there are tabs for Home, My Actions, Admin, Reporting, Manage Workflow, Manage Events, and Sample. The Admin tab is selected. The main area has two sections: 'Manage User Roles' and 'Edit Tenant Mapping'. In 'Manage User Roles', there is a table with columns for Role Name (TechcelloIndia), Description (Tenant Admin), Manage Tenants (button), and Remove (button). Below this is a table for 'Edit Tenant Mapping' with columns for Role Name (Tenant Admin), Description (Tenant Admin Role), and Tenants. Under 'Tenants', three checkboxes are checked: TechcelloDelhi, TechcelloIndia, and TechcelloMumbai. At the bottom right are 'Cancel' and 'Save' buttons. The footer includes the 'techcello' logo and copyright information: Copyright © 2012 by techcello.com.

Figure 8-5 – Manage User Roles Screen

8.2.1 Assigning Privileges to roles

Assignable Privilege

You can define a set of privileges and assign them to a particular role by using RoleAssignablePrivilege. You have to insert Role-Privilege mapping from the back end in RoleAssignablePrivilege table. The schema for RoleAssignablePrivilege table is given below.

Column Name	Data Type	Allow Nulls
RoleAssignablePrivilege_Id	Uniqueidentifier	False
RoleAssignablePrivilege_RoleId	Varchar (150)	False
RoleAssignablePrivilege_Privilegeld	Varchar (50)	False
RoleAssignablePrivilege_CreatedBy	Varchar (50)	False
RoleAssignablePrivilege_CreatedOn	DateTime	False
RoleAssignablePrivilege_UpdatedBy	Varchar (50)	True

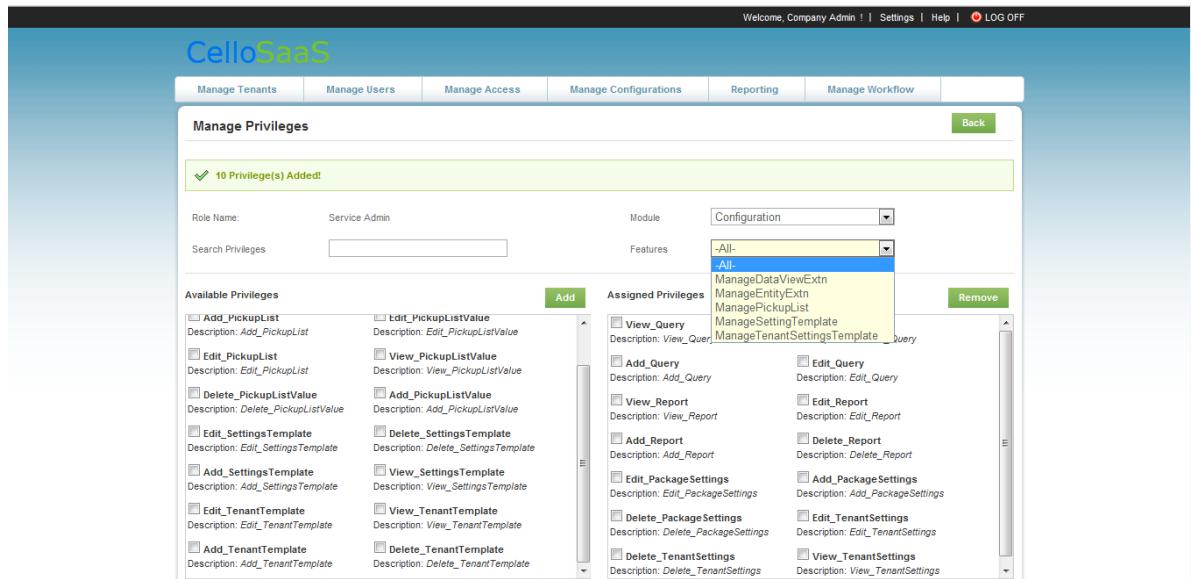
RoleAssignablePrivilege_UpdatedOn	DateTime	True
RoleAssignablePrivilege_Status	Bit	False

If the roles are not mapped to a service then intersection of assignable privileges and tenant license privileges will be displayed.

If the roles are mapped to one or more services, then intersection of assignable privileges, service based privileges and tenant license privileges will be displayed.

Employee

Employee role is not mapped with any of the service. Hence you will get the intersection of assignable privileges (If it is defined) and tenant license privileges in manage privilege page.



The screenshot shows the 'Manage Privileges' page for the 'Service Admin' role. The 'Available Privileges' section lists various service-based privileges like 'Add_PickupList', 'Edit_PickupList', etc. The 'Assigned Privileges' section shows the intersection of these with tenant license privileges, with a dropdown menu open for 'Features' showing options like '-All-' and 'All'. A green message bar at the top indicates '10 Privilege(s) Added!'.

Figure 8-6 – Manage Privileges Screen

Userservice

UserService role is mapped with User service. Hence you will get the intersection of assignable privileges (If it is defined), service based privileges and tenant license privileges in manage privilege page.

Tenant	siemens		
Entity	View Tenant Scope	Edit Tenant Scope	Delete Tenant Scope
Package	Default Tenant Scope	Default Tenant Scope	Default Tenant Scope
User	Default Tenant Scope	Default Tenant Scope	Default Tenant Scope
Tenant	Default Tenant Scope	Default Tenant Scope	Default Tenant Scope
Role	Default Tenant Scope	Default Tenant Scope	Default Tenant Scope
TenantSettingsTemplate	Default Tenant Scope	Default Tenant Scope	Default Tenant Scope
SettingsTemplate	Selective Tenants	Immediate Child Tenants	Default Tenant Scope

Select Tenant Scope

Figure 8-7 – Manage Entity Tenant Scope Screen

Service Administrator

Service administrator is a user who has the role “Service Admin”. This user will be created by Product admin. Service admin has the following privileges.

Creating, managing Global roles and mapping with his services.

Assigning Privileges to the service-mapped roles.

Creating a user and mapping roles of his services.

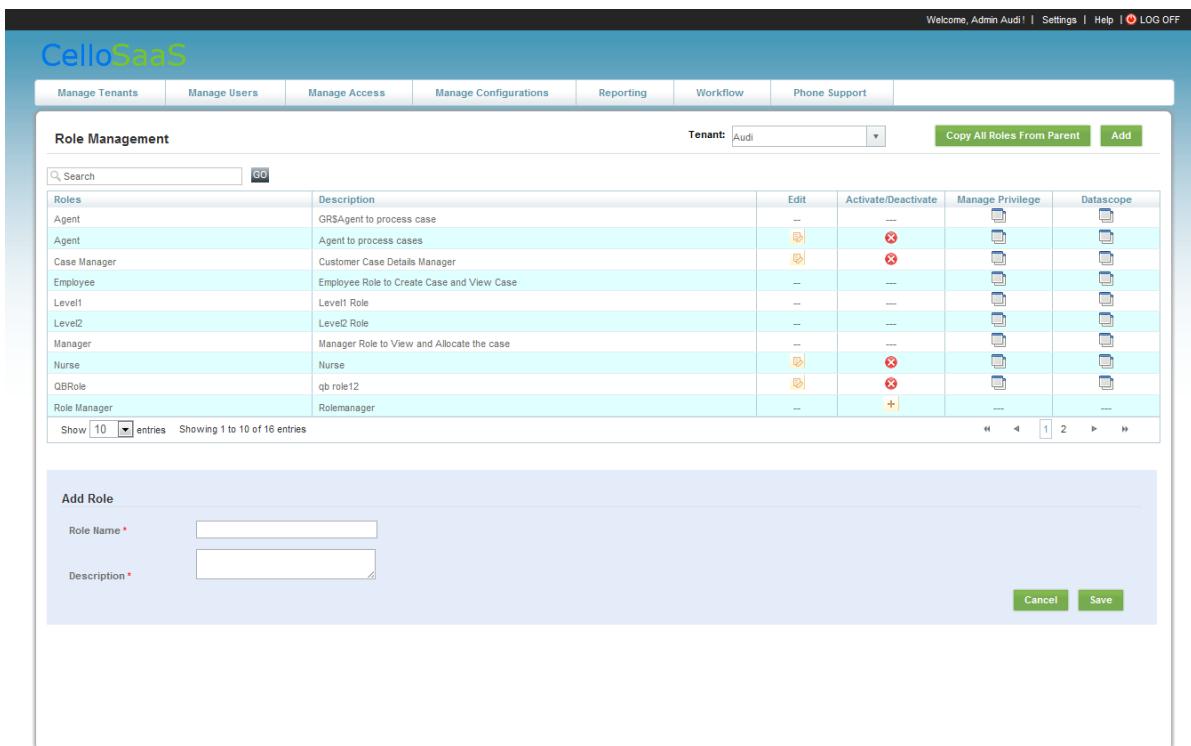
Creating, managing Global roles and mapping services

Roles which are mapped to the services will be displayed fro service admin in Role details page

Service administrators can create global role with service mapping. The user cannot create tenant based role or global role without service mapping.

Once you check the Global Role checkbox, services which are mapped to logged-in Service Admin will be displayed.

Likewise the user cannot change the service mapping for the created roles. The description of the roles can only be modified.



Roles	Description	Edit	Activate/Deactivate	Manage Privilege	DataScope
Agent	GRSAgent to process case	—	—		
Agent	Agent to process cases				
Case Manager	Customer Case Details Manager				
Employee	Employee Role to Create Case and View Case	—	—		
Level1	Level1 Role	—	—		
Level2	Level2 Role	—	—		
Manager	Manager Role to View and Allocate the case	—	—		
Nurse	Nurse				
QBRole	qb role12				
Role Manager	Rolemanager	—		—	—

Figure 8-8 – Role Management Screen

Assigning Privileges to the service-mapped roles

In manage privilege page, for service admin, intersection of assignable privileges, service based privilege (services which are mapped to the role) and tenant license privileges will be displayed.

Creating user and mapping roles to services

If the logged in user is a service administrator, then the roles which are mapped to the services will be displayed. The user cannot assign tenant based roles to the users.

If the logged in user is product administrator, then all tenant based roles and global roles will be displayed.

If the logged in user is other than service administrator and product administrator, then the roles available to the tenant license and tenant based roles will be displayed.

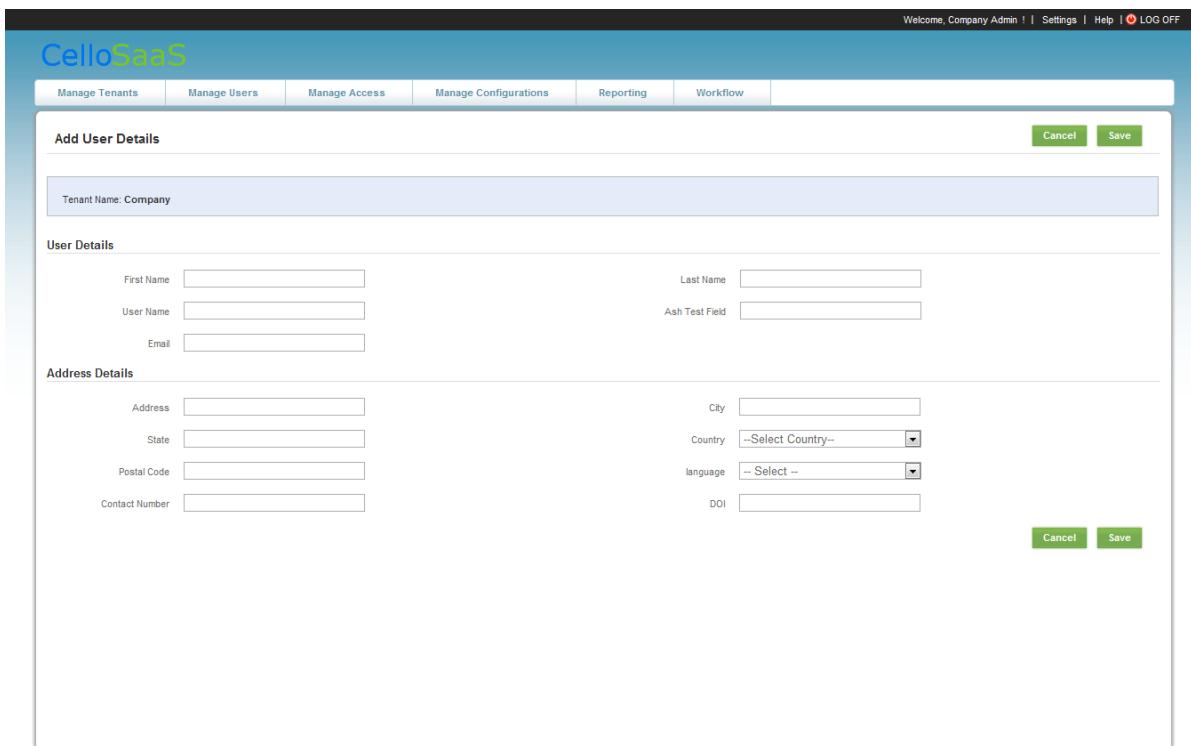


Figure 8-9 – Add User Details Screen

Service Managements

You can also consume following services for Role-Service mapping which will be available in CelloSaaS.Services at CelloSaaS.Services.LicenseManagement.LicenseService.

```

///<summary>
/// This method is used to get service details for given roleId.
///</summary>
///<param name="roleId"></param>
///<returns></returns>
Dictionary<string, Service> GetServicesByRoleId(string roleId);
///<summary>
/// Gets the service details for given roleIds.
///</summary>
///<param name="roleIds"></param>
///<returns></returns>
Dictionary<string, Service> GetServicesByRoleIds(string[] roleIds);
    ///<summary>
    /// This method is used to update role service.
    ///</summary>
    ///<param name="roleId"></param>
    ///<param name="serviceIds"></param>
    void UpdateRoleServices(string roleId, string[] serviceIds);
    ///<summary>
    /// This method is used to delete role services.
    ///</summary>
    ///<param name="roleId"></param>
    ///<param name="serviceIds"></param>
    void DeleteRoleServices(string roleId, string[] serviceIds);

```

You can also consume the following services to get role based on service mapping which will be available in CelloSaaS.Services at CelloSaaS.Services.AccessControlManagement.RoleService.

```
///<summary>
/// This method is used to get all roles for the given serviceld
///</summary>
///<param name="serviceld">Service Id</param>
///<returns>List of role details</returns>
Dictionary<string, Role> GetRolesByServiceld(string serviceld);
///<summary>
/// This method is used to get all roles for the given serviceIds
///</summary>
///<param name="serviceIds">Service Ids</param>
///<returns>List of role details</returns>
Dictionary<string, Role> GetRolesByServiceIds(string[] serviceIds);
///<summary>
/// Gets the roles by service admin user.
///</summary>
///<param name="serviceAdminUserId">The service admin user id.</param>
///<returns></returns>
Dictionary<string, Role> GetRolesByServiceAdminUser(string serviceAdminUserId);
///<summary>
/// Gets the global roles by tenant.
///</summary>
///<param name="tenantId">The tenant id.</param>
///<returns></returns>
List<Role> GetGlobalRolesByTenant(string tenantId);
```

You can also consume the following services to get privilege based on role-service mapping which will be available in CelloSaaS.Services at CelloSaaS.Services.AccessControlManagement.PrivilegeService.

```
///<summary>
/// This method is used to get privileges for given serviceIds.
///</summary>
///<param name="serviceIds"></param>
///<returns></returns>
Dictionary<string, Privilege> GetPrivilegesByServiceIds(string[] serviceIds);
///<summary>
/// Get assignable privileges for given role.
///</summary>
///<param name="roleId"></param>
///<returns></returns>
Dictionary<string, Privilege> GetDefaultAssignablePrivilegesForRole(string roleId);
///<summary>
/// Get AssignablePrivilege of current user for given role.
///</summary>
///<param name="roleId"></param>
///<returns></returns>
Dictionary<string, Privilege> GetCurrentUsersAssignablePrivilegesForRole(string roleId);
```

8.3 Privilege Management

Privileges

Privileges are used for access control.



Access privileges determine which application objects a user can browse or edit, which objects appear in search results, and which can be accessed by the user. CelloSaaS allows users to do any action only after validating their Access privilege. Any Business entity will have the basic CRUD privileges in order to manipulate on, in addition to basic privileges extra privileges can also be added.

Example

Object: Employee

Privilege: View, Add, Update, Delete

Other Privilege: Search, Sort

Privileges are maintained in the Privileges table. Privileges can be either free form Privileges or Entity Action based Privileges such as “View_Employee”, “Edit-Employee”. All the entity action based privileges should follow the naming convention “{Action}_{EntityName}”. Entity Action Privileges are used to restrict the access to various actions of an entity, for example, “View_Employee” permission is required to view the records of entity –“Employees”. “Edit_Employee” permission is required to edit the records of entity –“Employees”. Developer Usage”Database:

All the Entity Action Privileges should be mapped with Entity in the database. Need to put entry in EntityPrivileges Table.

EntityId – Entity Identifier which has been created in entity table

PrivilegeId – This has been created in Entity table

Role Privilege

As mentioned before every role can be mapped to one or many privileges. Role privilege details are stored in the “RolePrivilege” table. To manage the roles through code, following method needs to be used:

```
To add role privilege  
void AddPrivilegesToRole(string tenantIdentifier  
List<RolePrivilege> rolePrivileges) – Add more than one privilege to role.  
string AddPrivilegeToRole(string tenantIdentifier  
RolePrivilege rolePrivilege) – Add privilege to role.
```

Add or update role privilege: If a privilege is available to a role, then update the role privilege or add privilege to role.

```
void SavePrivilegesToRole(string tenantIdentifier  
List<RolePrivilege> rolePrivileges)
```

Roles and Privileges combination are unique for each tenant. Delete role privilege by using the following method.

```
void DeletePrivilegesForRole(string tenantId, string roleId, string[] privilegeIds) – Delete more  
privileges from role.  
void DeleteRolePrivilege(string rolePrivilegeId) – Delete privilege from role.
```

When a user logs in, all his privileges are attached to his identity which can be obtained using the property *CelloSaaS.Library.UserIdentity.Privileges*.

8.4 Data Access Management

CelloSaaS providers should be in a position to provide their tenant the ability to configure their access control policy at DataLevel. CelloSaaS provides the “Data Access Management” feature which allows developers to provide support to their tenants easily. “Datascope” feature is used for data level access management. Datascope defines the visibility of data. It helps to restrict the user access at data level. When a role is assigned for privileges, it is given with respect to a datascope; the role possesses that privilege for all the data within the given datascope.

Below is an example that gives a clear picture of datascope.

Consider Employee management application as example, the following datascope are defined:

Privilege	Description
My Team	The user can access only his/her team members details.
My Department	The user can access only his department members details.
All Data	The user can access all Employee Details. This is more or less same as without mapping the datascope.
Nil	The user cannot access any Employee details.

Developer Usage: Once you define a datascope and map it with privileges, that privilege is demanded in the code; CelloSaaS appends the datascope query with your actual query in the DAL and provides back the result with filtered condition. Mostly datascope query will be appended with your fetch and search query.

Database: Separate logical code for defining datascope is not required. You will only have to define the datascope from the back end. To define a datascope for a business model object, you need to put entry in the following two tables.

DatascopeNameVarchar(100): Name of the datascope

SelectQueryVarchar(255): Filter query which is going to be appended with your fetch or search query

ConditionVarchar(255): If required, you can give condition to your database query.

AccessLevelSmallint: Sets the priority of the datascope. Same privilege may be mapped with different datascopes in different roles. If an user possesses two roles, then that particular privilege will get two datascopes. At that time, CelloSaaS will consider the datascope which has high priority access level.

EntityDataScope

DatascopeIdVarchar(100): This has been created in Datascope table.

EntityIdVarchar(200): EntityIdentifier which has been created in entity table.

BridgeConditionVarchar(255): This is the conjunction property to append your fetch or search query with Datascope query.

To access the service, you have to demand the privilege by using “CelloSaaS.Library. Context” which is provided by CelloSaaS.Context has the following methods.

```

///<summary>
/// This method is used to create PrivilegeContext instance and that will be add in
HttpContext.Current.Items with "Privilege" as key
///</summary>
///<param name="privilegeName"></param>
void SetPrivilegeContext(string privilegeName)
///<summary>
/// This method is used to get the PrivilegeContext instance which is currently present.
///</summary>
///<returns>PrivilegeContext which has demanded privilege</returns>
CelloSaaS.Library.PrivilegeContext GetCurrentPrivilegeContext()

```

Usually privilege will be demanded from the controller before calling Proxy methods by using SetPrivilegeContext method.

Example

```
CelloSaaS.Library.Context.SetPrivilegeContext("Edit_EmployeeDetails");
```

In services, current privilege will be taken from HttpContext by using GetCurrentPrivilegeContext method.

Example

```

string currentPrivilege = string.Empty;
PrivilegeContext privilegeContext =
    Context.GetCurrentPrivilegeContext();
if (privilegeContext != null)
    currentPrivilege = privilegeContext.PrivilegeName;

```

To fetch and search

```

///<summary>
/// To set dataScope and entityBasedDataScope in out parameter for given entityIdentifier and
permission.
///</summary>
///<param name="entityIdentifier"> EntityIdentifier of business object </param>
///<param name="permissionName">Privilege name</param>
///<param name="dataScope">Datascope as out parameter</param>
///<param name="entityDataScope">EntityDatascope as out parameter</param>
void SetDataScopes(string entityIdentifier, string permissionName, outDataScope dataScope,
outEntityBasedDataScope entityDataScope)

```

To create “fetch request” instance and call the fetch method

```

///<summary>
/// Request object which has Identifier,datascope,entityBaseDataScope.
///</summary>
///<param name="identifier"> Primary Id of the record</param>
///<param name="datascope"> Datascope for demanded privilege</param>
///<param name="entityBasedDatascope"> EntityBasedDatascope of the entity for demanded
privilege</param>
CelloSaaS.DAL.DataFetchRequest (string identifier, DataScope dataScope, EntityBasedDataScope
entityBasedDataScope)

```

To create “search request” instance and call the search method

```

///<summary>
/// Request object which has list of Identifier,dataScope,entityBaseDataScope.
///</summary>
///<param name="identifiers"> List of primary Ids which are to be fetched</param>
///<param name="dataScope"> Datascope for demanded privilege</param>
///<param name="entityBasedDataScope"> EntityBasedDatascope of the entity for demanded
privilege</param>
cellosaaa.DAL.DataSearchRequest (string[] identifiers, DataScope dataScope,
EntityBasedDataScope entityBasedDataScope)

```

Create “update request” instance and call the update method

```

///<summary>
/// Request object which has identifier and business object to be updated
///</summary>
///<param name="identifier"> Primary Id of the record</param>
///<param name="entity"> Entity object which is going to be updated</param>
CelloSaaS.DAL.DataUpdateRequest (string identifier, BaseEntity entity)

```

To create “delete request” instance and call the delete method

```

///<summary>
/// Request object which has identifier and business object to be deleted
///</summary>
///<param name="identifier"> Primary Id of the record</param>
///<param name="entity"> Entity object which is going to be deleted</param>
CelloSaaS.DAL.DataDeleteRequest (string identifier, BaseEntity entity)

```

To update and delete

```

DataUpdateRequest dataUpdateRequest = new
    DataUpdateRequest(employeeDetails.Identifier, employeeDetails);
employeeDetailsDAL.Update(dataUpdateRequest);
You can validate the access before performing the fetch and search using the following method
if (!base.ValidateAccess(entityAction, employeeDetails))
{
    string messsage = string.Format("Permission to update the given
record is denied.");
    throw new UnauthorizedAccessException(messsage);
}

```

DAL



In DAL, to implement the actions like DoCreate, DoUpdate, DoDelete, DoSearch you will have to use CreateQuery, FetchQuery, DeleteQuery, SearchQuery to build the queries.

```

///<summary>
/// It is used to build a fetch query with Datascope filter query
///</summary>
///<param name="connectionStringName"></param>
///<param name="dataScope"></param>
///<param name="entityBasedDataScope"></param>
CelloSaaS.DAL.FetchQuery(string connectionStringName, DataScope dataScope,
EntityBasedDataScope entityBasedDataScope)
///<summary>
/// It is used to build a inert query

```

```

///</summary>
///<param name="connectionStringName"></param>
CelloSaaS.DAL.InsertQuery (string connectionString)
///<summary>
/// It is used to build a update query
///</summary>
///<param name="connectionStringName"></param>
CelloSaaS.DAL.UpdateQuery (string connectionString)
///<summary>
/// It is used to build a delete query
///</summary>
///<param name="connectionStringName"></param>
CelloSaaS.DAL.UpdateQuery (string connectionString)

```

8.5 Page Level Access Management

Page level access is obtained by providing the configuration in config file “EntityPermissions.config”. All the URL’s can be mapped to the required access control rule which is then demanded automatically whenever that page/action is called. Automatically menu integrates with this and turns off the menus to which the page’s access rule is not met.

Access rule is a combination of both privileges and roles. For example, Access Rule “P:View_Employee And P>Edit_Employee” states that the user should possess both “View_Privilege” and “Edit_Privilege” to access the page.

The configuration should be similar to the following:

```

<EntityPermission>
<EntityCategory>
<addname="UI">
<Entity>
<addname="/tenant/**"AuthorizationRule="R:GR$Product_Admin"/>
<addname="/tenant/tenantsettings"AuthorizationRule="R:GR$Tenant_Admin"/>
<addname="/roles/**"AuthorizationRule="R:GR$Tenant_Admin"/>
<addname="/user/**"AuthorizationRule="R:GR$Tenant_Admin"/>
<addname="/data/**"AuthorizationRule="R:GR$Tenant_Admin"/>
</Entity>
</EntityCategory>
</EntityPermission>

```

In Entity tag, you will have to mention menus url or actions in name attribute. And authorizationRule will be used to mention access in the form Role or Privilege or Identity. You will have to give authorizationRule in the following format:

For Role, R:RoleName. Example: R:GR\$Product_Admin

For Privilege, P:PrivilegeName Example: P:AddEmployee

For Identity, I:Identity Example: I:admin@company.com

The permission can be constructed as a complex expression containing AND, OR, NOT.

Example

AuthorizationRule=R:GR\$Product_Admin OR R:GR\$Tenant_Admin
 You can also combine permissions and roles in an expression.

8.6 Field Level Access Management

Using CelloSaaS, developers can easily provide support to control the access at a field level in their product.

Administration: Field Privileges can be set by using Manage Role-Privilege screen which is provided by CelloSaaS.

Each entity has set of fields.

Entity fields have Edit and View privileges.

We can set the privileges for entity fields for available roles.

Tenant Field List				Back	Save
Field Name	Visibility		Editable	Back	Save
AddressId	<input checked="" type="checkbox"/> -All-		<input type="checkbox"/> -All-		
TenantCode	<input checked="" type="checkbox"/> -All-		<input type="checkbox"/> -All-		
ContactId	<input checked="" type="checkbox"/> -All-		<input type="checkbox"/> -All-		
Description	<input checked="" type="checkbox"/> -All-		<input type="checkbox"/> -All-		
IsProductAdmin	<input type="checkbox"/> -All-		<input type="checkbox"/> -All-		
TenantName	<input type="checkbox"/> -All-		<input type="checkbox"/> -All-		
Status	<input type="checkbox"/> -All-		<input type="checkbox"/> -All-		
URL	<input type="checkbox"/> -All-		<input type="checkbox"/> -All-		
Website	<input type="checkbox"/> -All-		<input type="checkbox"/> -All-		
country	<input type="checkbox"/> -All-		<input type="checkbox"/> -All-		
tst	<input type="checkbox"/> -All-		<input type="checkbox"/> -All-		

Figure 8-10 - Field level Security

Each field has two privileges:

Visibility

Edit-ability

Data scope mapped with privileges will also be applied in field level access. It is not required to enter the Field Level Privileges. If the privilege does not exist, CelloSaaS will put the entry while assigning the privileges for a user. CelloSaaS follows a naming convention for privileges name.

For Visibility: ViewField_FieldIdentifier

Example: ViewField_FirstName

For Edit-ability: EditField_FieldIdentifier

Example: EditField_FirstName

If you do not set the privileges to a field, CelloSaaS will apply parent entity privileges. To bring out the field level privilege feature, you do not need to write any specific code. Dataview table and cello Grid will take care of field level access by using FieldIdentifier which have been given.

8.7 User Entity Access Management

User Entity Permission is used to provide the access at data level. Using this, one can set the Permissions and Privileges for the entity reference against the particular User or Role. A developer can set the Permissions and Privileges in three following ways:

- Open To All Users and Roles (Global Permission)
- For Particular User
- For Particular Role

Models

There are following two models used in User Entity Permission

- `UserEntityPermission`
 - This model contains the User Entity Permission details with `EntityId`, `Referenceld`, `UserId`, `Roleld` and others properties.
- `UserEntityOtherPrivilege`
 - This is used to save the other privileges for the user entity permission.

```
namespace CelloSaaS.Model.AccessControlManagement;
public class UserEntityPermission
{
    /// <summary>
    /// Gets or sets the Id.
    /// </summary>
    /// <value>
    /// The Id.
    /// </value>
    public string Id{ get; set; }

    /// <summary>
    /// Gets or sets the EntityId.
    /// </summary>
    /// <value>
    /// The EntityId.
    /// </value>
    public string EntityId{ get; set; }

    /// <summary>
    /// Gets or sets the RefernceId.
    /// </summary>
    /// <value>
    /// The RefernceId.
    /// </value>
    public string ReferenceId{ get; set; }

    /// <summary>
    /// Gets or sets the ReferenceName.
    /// </summary>
```

```
    /// <value>
    /// The ReferenceName.
    /// </value>
    public string ReferenceName{ get; set; }

    /// <summary>
    /// Gets or sets the FieldIdentifier.
    /// </summary>
    /// <value>
    /// The FieldIdentifier.
    /// </value>
    public string FieldIdentifier{ get; set; }

    /// <summary>
    /// Gets or sets the UserId.
    /// </summary>
    /// <value>
    /// The UserId.
    /// </value>
    public string UserId{ get; set; }

    /// <summary>
    /// Gets or sets the role id.
    /// </summary>
    /// <value>
    /// The role id.
    /// </value>
    public string RoleId{ get; set; }

    /// <summary>
    /// Gets or sets the role ids.
    /// </summary>
    /// <value>
    /// The role ids.
    /// </value>
    public List<string> RoleIds{ get; set; }

    /// <summary>
    /// Gets or sets the TenantId.
    /// </summary>
    /// <value>
    /// The TenantId.
    /// </value>
    public string TenantId{ get; set; }

    /// <summary>
    /// Gets or sets the View.
    /// </summary>
    /// <value>
    /// The View.
    /// </value>
    public bool? ViewPermission{ get; set; }

    /// <summary>
    /// Gets or sets the Add.
    /// </summary>
    /// <value>
```

```
    ///> The Add.
    ///> </value>
    public bool? AddPermission{ get; set; }

    ///> <summary>
    ///> Gets or sets the Edit.
    ///> </summary>
    ///> <value>
    ///> The Edit.
    ///> </value>
    public bool? EditPermission{ get; set; }

    ///> <summary>
    ///> Gets or sets the Delete.
    ///> </summary>
    ///> <value>
    ///> The Delete.
    ///> </value>
    public bool? DeletePermission{ get; set; }

    ///> <summary>
    ///> Gets or sets the User Entity Other Permissions.
    ///> </summary>
    ///> <value>
    ///> The UserEntityOtherPermissions.
    ///> </value>
    public List<UserEntityOtherPrivilege> UserEntityOtherPrivileges { get; set; }

    ///> <summary>
    ///> Gets or sets the CreatedBy.
    ///> </summary>
    ///> <value>
    ///> The CreatedBy.
    ///> </value>
    public string CreatedBy{ get; set; }

    ///> <summary>
    ///> Gets or sets the CreatedOn.
    ///> </summary>
    ///> <value>
    ///> The CreatedOn.
    ///> </value>
    public DateTime CreatedOn{ get; set; }

    ///> <summary>
    ///> Gets or sets the UpdatedBy.
    ///> </summary>
    ///> <value>
    ///> The UpdatedBy.
    ///> </value>
    public string UpdatedBy{ get; set; }

    ///> <summary>
    ///> Gets or sets the UpdatedOn.
    ///> </summary>
    ///> <value>
```



```
    ///> The UpdatedOn.
    ///> </value>
    public DateTime? UpdatedOn { get; set; }

    ///> <summary>
    ///> Gets or sets the Status.
    ///> </summary>
    ///> <value>
    ///> The Status.
    ///> </value>
    public bool Status{ get; set; }
}

namespace CelloSaaS.Model.AccessControlManagement;
public class UserEntityOtherPrivilege
{
    ///> Gets or sets the other permission id.
    ///> </summary>
    ///> <value>
    ///> The other permission id.
    ///> </value>
    ///> <summary>
    public string OtherPermissionId { get; set; }

    ///> <summary>
    ///> Gets or sets the user entity permission id.
    ///> </summary>
    ///> <value>
    ///> The user entity permission id.
    ///> </value>
    public string UserEntityPermissionId { get; set; }

    ///> <summary>
    ///> Gets or sets the privilege.
    ///> </summary>
    ///> <value>
    ///> The privilege.
    ///> </value>
    public string Privilege { get; set; }

    ///> <summary>
    ///> Gets or sets the CreatedBy.
    ///> </summary>
    ///> <value>
    ///> The CreatedBy.
    ///> </value>
    public string CreatedBy { get; set; }

    ///> <summary>
    ///> Gets or sets the CreatedOn.
    ///> </summary>
    ///> <value>
    ///> The CreatedOn.
    ///> </value>
    [DataMember]
    public DateTime CreatedOn { get; set; }
```

```

    ///<summary>
    /// Gets or sets the UpdatedBy.
    /// </summary>
    /// <value>
    /// The UpdatedBy.
    /// </value>
    public string UpdatedBy { get; set; }

    ///<summary>
    /// Gets or sets the UpdatedOn.
    /// </summary>
    /// <value>
    /// The UpdatedOn.
    /// </value>
    public DateTime? UpdatedOn { get; set; }

    ///<summary>
    /// Gets or sets the Status.
    /// </summary>
    /// <value>
    /// The Status.
    /// </value>
    public bool Status { get; set; }
}

```

User Entity Permission Services

```

using CelloSaaS.ServiceContracts.AccessControlManagement;
public interface IUserEntityPermissionService
{
    ///<summary>
    /// This method is used to add the User Entity Permission details
    /// </summary>
    ///<param name="userEntityPermission">The user entity permission.</param>
    ///<returns>
    /// Id
    /// </returns>
    string AddUserEntityPermission(UserEntityPermission userEntityPermission);

    ///<summary>
    /// update the UserEntityPermission details
    /// </summary>
    ///<param name="userEntityPermission">The user entity permission.</param>
    void UpdateUserEntityPermission(UserEntityPermission userEntityPermission);

    ///<summary>
    /// delete the UserEntityPermission details by id
    /// </summary>
    ///<param name="id">The id.</param>
    void DeleteUserEntityPermission(string id);

    ///<summary>
    /// Get all UserEntityPermission details for the tenantId.
    /// </summary>
    ///<param name="tenantId">The tenant id.</param>
    ///<returns>

```



```
/// Dictionary
/// </returns>
Dictionary<string, UserEntityPermission> GetUserEntityPermissionsByTenantId(string tenantId);

/// <summary>
/// Get all UserEntityPermission details for the search condition.
/// </summary>
/// <param name="userEntityPermissionSearchCondition">The user entity permission search condition.</param>
/// <returns>
/// Dictionary
/// </returns>
Dictionary<string, UserEntityPermission> SearchUserEntityPermissionDetails(UserEntityPermissionSearchCondition userEntityPermissionSearchCondition);

/// <summary>
/// This method is used to get the UserEntityPermission count
/// </summary>
/// <param name="userEntityPermissionSearchCondition">The user entity permission search condition.</param>
/// <returns>
/// int
/// </returns>
int SearchUserEntityPermissionDetailsCount(UserEntityPermissionSearchCondition userEntityPermissionSearchCondition);

/// <summary>
/// Saves the user entity other privileges.
/// </summary>
/// <param name="userEntityPermission">The user entity permission.</param>
void SaveUserEntityOtherPrivileges(UserEntityPermission userEntityPermission);

/// <summary>
/// This method is used to get ReferenceId by Entity
/// </summary>
/// <param name="userId">User Id</param>
/// <param name="entityId">Entity Id</param>
/// <param name="operation">Entity Operation<see cref="EntityOperation"/></param>
>
/// <returns>
/// Reference Ids
/// </returns>
string[] GetReferenceIdByEntity(string userId, string entityId, EntityOperation operation);

/// <summary>
/// This method is used to get ReferenceId by Entity
/// </summary>
/// <param name="userId">User Id</param>
/// <param name="entityIds">Entity Ids</param>
/// <param name="operation">Entity Operation<see cref="EntityOperation"/></param>
>
/// <returns>
/// Reference Ids
/// </returns>
```



```
Dictionary<string, string[]> GetReferenceIdByEntities(string userId, string[] en  
tityIds, EntityOperation operation);  
  
/// <summary>  
/// This method will be used to get reference Ids.  
/// </summary>  
/// <param name="userId">User Id</param>  
/// <param name="entityId">Entity Id</param>  
/// <param name="operation">Entity Operation<see cref="EntityOperation"/></param  
>  
/// <returns>  
/// Dictionary of Reference Ids  
/// </returns>  
Dictionary<string, string[]> GetReferenceIdByOperations(string userId, string en  
tityId, EntityOperation[] operation);  
  
/// <summary>  
/// This method is used to get ReferenceId by Privilege  
/// </summary>  
/// <param name="userId">User Id</param>  
/// <param name="entityId">Entity Id</param>  
/// <param name="privilege">Privilege</param>  
/// <returns>  
/// Reference Ids  
/// </returns>  
string[] GetReferenceIdByPrivilege(string userId, string entityId, string privil  
ege);  
  
/// <summary>  
/// This method is used to get the reference id by privileges.  
/// </summary>  
/// <param name="userId">The user id.</param>  
/// <param name="entityId">The entity id.</param>  
/// <param name="privileges">The privileges.</param>  
/// <returns>  
/// List of Reference Ids  
/// </returns>  
Dictionary<string, string[]> GetReferenceIdByPrivileges(string userId, string en  
tityId, string[] privileges);  
  
/// <summary>  
/// Checks the user entity permission by other permission.  
/// </summary>  
/// <param name="entityId">The entity id.</param>  
/// <param name="userId">The user id.</param>  
/// <param name="referenceId">The reference id.</param>  
/// <param name="privilege">The privilege.</param>  
/// <returns></returns>  
bool CheckUserEntityPermissionByOtherPrivilege(string entityId, string userId, s  
tring referenceId, string privilege, string tenantId);  
  
/// <summary>  
/// Checks the user entity permission by other permission.  
/// </summary>  
/// <param name="entityId">The entity id.</param>  
/// <param name="userId">The user id.</param>  
/// <param name="referenceId">The reference id.</param>
```

```

/// <param name="privileges">The privileges.</param>
/// <returns></returns>
Dictionary<string, bool> CheckUserEntityPermissionByOtherPrivileges(string entit
yId, string userId, string referenceId, string[] privileges, string tenantId);

/// <summary>
/// Checks the user entity permission by other permission.
/// </summary>
/// <param name="entityId">The entity id.</param>
/// <param name="userId">The user id.</param>
/// <param name="referenceId">The reference id.</param>
/// <param name="entityOperation">The entity operation.</param>
/// <returns></returns>
bool CheckUserEntityPermissionByEntityOperation(string entityId, string userId,
string referenceId, EntityOperation entityOperation, string tenantId);

/// <summary>
/// Checks the user entity permission by other permission.
/// </summary>
/// <param name="entityId">The entity id.</param>
/// <param name="userId">The user id.</param>
/// <param name="referenceId">The reference id.</param>
/// <param name="entityOperations">The entity operations.</param>
/// <returns></returns>
Dictionary<string, bool> CheckUserEntityPermissionByEntityOperations(string enti
tyId, string userId, string referenceId, EntityOperation[] entityOperations, str
ing tenantId);

}

```

User Entity Permission Scenarios:

Using this can set permissions and privileges to the data level to the user and role. Let's take *Document* as Entity and we have *Document1*, *Document2*, and *Document3* in a entity. These are all called entity reference. In this we can set permissions like Add, View, Edit, and Delete and other privileges like Search, Publish for the particular user or role.

To Create the User Entity Permission we have to use following services:

```

/// <summary>
/// This method is used to add the User Entity Permission details
/// </summary>
/// <param name="userEntityPermission">The user entity permission.</param>
/// <returns>
/// Id
/// </returns>
string AddUserEntityPermission(UserEntityPermission userEntityPermission);

```

To Edit the User Entity Permission we have to use following services:

```

/// <summary>
/// This method is used to update the User Entity Permission details
/// </summary>
/// <param name="userEntityPermission">The user entity permission.</param>
void UpdateUserEntityPermission(UserEntityPermission userEntityPermission);

```

To Delete the User Entity Permission we have to use following services:

```

/// <summary>
/// This method is used to delete the User Entity Permission details by id
/// </summary>

```



```
/// <param name="id">The id.</param>
void DeleteUserEntityPermission(string id);
```

To Search User Entity Permission we have to use following services:

```
/// <summary>
/// This method is used to get all UserEntityPermission details for the search
condition.
/// </summary>
/// <param name="userEntityPermissionSearchCondition">The user entity permission
search condition.</param>
/// <returns>
/// Dictionary<string, UserEntityPermission>
/// </returns>
Dictionary<string, UserEntityPermission> SearchUserEntityPermissionDetails(UserE
ntityPermissionSearchCondition userEntityPermissionSearchCondition);
```

To Add and Update User Entity Other Privilege we have to use following services:

```
/// <summary>
/// This method is used to save the user entity other privileges.
/// </summary>
/// <param name="userEntityPermission">The user entity permission.</param>
void SaveUserEntityOtherPrivileges(UserEntityPermission userEntityPermission);
```

APIs

The following APIs are used to get the referenceIds by giving the `userId`, `entityId`, `entityOperation`, and privileges.

```
/// <summary>
/// This method is used to get ReferenceId by Entity
/// </summary>
/// <param name="userId">User Id</param>
/// <param name="entityId">Entity Id</param>
/// <param name="operation">Entity Operation <see cref="EntityOperation" /></param>
/// <returns>
/// Reference Ids
/// </returns>
string[] GetReferenceIdByEntity(string userId, string entityId, EntityOperation operation);

/// <summary>
/// This method is used to get ReferenceId by Entity
/// </summary>
/// <param name="userId">User Id</param>
/// <param name="entityIds">Entity Ids</param>
/// <param name="operation">Entity Operation <see cref="EntityOperation" /></param>
/// <returns>
/// Reference Ids
/// </returns>
Dictionary<string, string[]> GetReferenceIdByEntities(string userId, string[] entityIds, EntityOperation operation);

/// <summary>
/// This method will be used to get reference Ids.
/// </summary>
/// <param name="userId">User Id</param>
/// <param name="entityId">Entity Id</param>
/// <param name="operation">Entity Operation <see cref="EntityOperation" /></param>
/// <returns>
/// Dictionary of Reference Ids
/// </returns>
Dictionary<string, string[]> GetReferenceIdByOperations(string userId, string entityId, EntityOperation[] operation);

/// <summary>
/// This method is used to get ReferenceId by Entity
/// </summary>
/// <param name="userId">User Id</param>
/// <param name="fieldId">Entity Id</param>
/// <param name="operation">Entity Operation <see cref="EntityOperation" /></param>
/// <returns>
/// Reference Ids
/// </returns>
string[] GetReferenceIdByField(string userId, string fieldId, EntityOperation operation);

/// <summary>
```

```

/// This method is used to get ReferenceId by Field
/// </summary>
/// <param name="userId">User Id</param>
/// <param name="fieldsIds">Entity Ids</param>
/// <param name="operation">Entity Operation <see cref="EntityOperation" /></param>
/// <returns>
/// Reference Ids
/// </returns>
Dictionary<string, string[]> GetReferenceIdByFields(string userId, string[] fieldsIds, EntityOperation operation);

/// <summary>
/// This method is used to get ReferenceId by Privilege
/// </summary>
/// <param name="userId">User Id</param>
/// <param name="entityId">Entity Id</param>
/// <param name="privilege">Privilege</param>
/// <returns>
/// Reference Ids
/// </returns>
string[] GetReferenceIdByPrivilege(string userId, string entityId, string privilege);

/// <summary>
/// This method is used to get the reference id by privileges.
/// </summary>
/// <param name="userId">The user id.</param>
/// <param name="entityId">The entity id.</param>
/// <param name="privileges">The privileges.</param>
/// <returns>
/// List of Reference Ids
/// </returns>
Dictionary<string, string[]> GetReferenceIdByPrivileges(string userId, string entityId, string[] privileges);

```

The following APIs are used to check the user entity permission for the data by using UserId, EntityId, Referenceld, EntityOperation, and Privileges

The **CheckUserEntityPermissionByOtherPrivilege** method is to check the permission with other privilege for the userId, entityId, referenceld, and tenantId. This will return the true/false if the permission has set for the reference.

```

/// <summary>
/// This method is used to check the user entity permission by other privilege.
/// </summary>
/// <param name="entityId">The entity id.</param>
/// <param name="userId">The user id.</param>
/// <param name="referenceId">The reference id.</param>
/// <param name="privilege">The privilege.</param>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
bool CheckUserEntityPermissionByOtherPrivilege(string entityId, string userId, string referenceId, string privilege, string tenantId);

```

The **CheckUserEntityPermissionByOtherPrivileges** method is to check the permissions with list of other privileges for the userId, entityId, referenceId, and tenantId. This will return the dictionary of true/false value for the privilege.

```
/// <summary>
/// This method is used to
check the user entity permission by other permissions.
/// </summary>
/// <param name="entityId">The entity id.</param>
/// <param name="userId">The user id.</param>
/// <param name="referenceId">The reference id.</param>
/// <param name="privileges">The privileges.</param>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
Dictionary<string, bool> CheckUserEntityPermissionByOtherPrivileges(string entit
yId, string userId, string referenceId, string[] privileges, string tenantId);
```

The **CheckUserEntityPermissionByEntityOperation** method is to check the permissions with entity operation for the userId, entityId, referenceId, and tenantId. This will return the true/false if the permission has set for the reference.

```
/// <summary>
/// This method is used to check the user entity permission by entity operation.
/// </summary>
/// <param name="entityId">The entity id.</param>
/// <param name="userId">The user id.</param>
/// <param name="referenceId">The reference id.</param>
/// <param name="entityOperation">The entity operation.</param>
/// <returns></returns>
bool CheckUserEntityPermissionByEntityOperation(string entityId, string userId,
string referenceId, EntityOperation entityOperation, string tenantId);
```

The **CheckUserEntityPermissionByEntityOperations** method is to check the permissions with array of entity operations for the userId, entityId, referenceId, and tenantId. This will return the dictionary of true/false for the entity operations.

```
/// <summary>
/// This method is used to
check the user entity permission by entity operations.
/// </summary>
/// <param name="entityId">The entity id.</param>
/// <param name="userId">The user id.</param>
/// <param name="referenceId">The reference id.</param>
/// <param name="entityOperations">The entity operations.</param>
/// <param name="tenantId">The tenant id.</param>
/// <returns></returns>
Dictionary<string, bool> CheckUserEntityPermissionByEntityOperations(string enti
tyId, string userId, string referenceId, EntityOperation[] entityOperations, str
ing tenantId);
```

Before going to start using the user entity permission some development things have to follow.

- The developer has to ensure the *Entity* (Class) must contain the *EntityDescriptor* properties such as **SchemaTableName**, **PrimaryKeyName**, and **DisplayColumnName**.
- The Entity table must contain the “**TenantId**” field
- The Display Column Name should not be Null field.
- The Display Column Name must be easily understandable column for ease.

Sample Entity

```
[EntityIdentifier(Name = "CaseDetail")]
[EntityDescriptor(
    PrimaryKeyName = "Id",
    DisplayName = "CaseNumber",
    SchemaTableName = "clm.CaseDetail",
    ExtnSchemaTableName = "CaseDetailExtn",
    IsExtensible = true,
    SchemaTableConnectionStringName = CelloSaaS.Library.DAL.Constants.ApplicationConnectionString)]
public class CaseDetail : BaseEntity
{
    public string TenantId { get; set; }
    public string CustomerId { get; set; }
    public int CaseNumber { get; set; }
    public CaseType CaseType { get; set; }
    public CaseStatus CaseStatus { get; set; }
    public CaseSubStatus CaseSubStatus { get; set; }
    public string DivisionId { get; set; }
    public string Division { get; set; }
    public CaseSource CaseSource { get; set; }
    public string Topic { get; set; }
    public string TopicTypeId { get; set; }
    public string TopicType { get; set; }
    public string TopicDetailId { get; set; }
    public string TopicDetail { get; set; }
    public string CategoryId { get; set; }
    public string Category { get; set; }
    public string ReasonId { get; set; }
    public string Reason { get; set; }
    public string CreatedBy { get; set; }
    public DateTime CreatedOn { get; set; }
    public string UpdatedBy { get; set; }
    public DateTime UpdatedOn { get; set; }
    public bool Status { get; set; }
}
```



Service Usage

Adding User Entity Permission for the data level

```
var userEntityPermission = new UserEntityPermission();
```

To add developer must set the UserId, RoleId, EntityId, ReferenceId, Permissions, OtherPrivileges and UpdatedBy, UpdatedOn, and Status as true.

```
userEntityPermission.CreatedBy = UserIdentity.UserId;
userEntityPermission.CreatedOn = DateTimeHelper.GetDatabaseDateTime();
userEntityPermission.UpdatedOn = true;
```

```
UserEntityPermissionProxy.AddUserEntityPermission(userEntityPermission);
```

For Update developer must set the UserId, RoleId, EntityId, ReferenceId, Permissions, OtherPrivileges and UpdatedBy, UpdatedOn, and Status as true.

```
userEntityPermission.Id = formCollection["UserEntityPermissionId"].ToString();
userEntityPermission.UpdatedBy = UserIdentity.UserId;
userEntityPermission.UpdatedOn = DateTimeHelper.GetDatabaseDateTime();
userEntityPermission.UpdatedOn = true;
```

```
UserEntityPermissionProxy.UpdateUserEntityPermission(userEntityPermission);
```

For Searching the specific User Entity Permission, can call as below. This will return Dictionary of UserEntityPermission.

```
UserEntityPermission userEntityPermission =
    UserEntityPermissionProxy.SearchUserEntityPermissionDetails(
        new UserEntityPermissionSearchCondition()
    {
        UserId = userId,
        RoleId = roleId,
        EntityId = entityId,
        ReferenceId = referenceId,
        TenantId = UserIdentity.TenantID,
        Status = true
    });
});
```

To get the reference lists for the entity the developer has to call the following service proxy as below.

```
Dictionary<string, string> referenceIds =
    DataManagementProxy.GetEntityKeyValuePair(entityId, UserIdentity.TenantID);
```

This service will return the Entity's **PrimaryKeyName** as Key and **DisplayColumnName** as Value in a dictionary.

To call the check user entity permission access has to call following service as below.

```
If(CelloSaaS.ServiceProxies.AccessControlManagement.UserEntityPermissionProxy.CheckUserEntityPermissionByEntityOperation(col.EntityIdentifier, userIdentifier, c
```



```
o1.Identifier, CelloSaaS.Model.EntityOperation.Update, tenantIdentifier) ==  
true){  
  
    //The true condition goes here.  
  
}
```

Screenshots

For User Entity Permission page click [Admin->Access Management ->User Entity Permission](#)

Select Entity List and choose the Entity Reference Select List, to set the permissions and privileges.

If you want to create the *Open to all* permission for the entity reference we can set without choosing the User and Role for the reference.

If you want to set permissions to the specific User or the Role, have to select the user from the user select list or the role select list.

Once we set the Permissions and Privileges for the entity reference click on *Add* Button to set the permission.

Entity	<input type="text" value="CaseDetail"/>	Entity Reference	<input type="text" value="2"/>	
User	<input type="text" value="Select User"/>	OR	Role	<input type="text" value="Select Role"/>
Permissions <input checked="" type="checkbox"/> View <input checked="" type="checkbox"/> Add <input checked="" type="checkbox"/> Edit <input checked="" type="checkbox"/> Delete				
Other Privileges <input type="text"/> +				
<input checked="" type="checkbox"/> Save Add				

Once added the user entity permission can able to update the user entity permission by selecting the Entity Reference, and User / Role.

Manage User Entity Permission

Entity	<input type="text" value="CaseDetail"/>	Entity Reference	<input type="text" value="4"/>	
User	<input type="text" value="Select User"/>	OR	Role	<input type="text" value="Select Role"/>
Permissions <input checked="" type="checkbox"/> View <input checked="" type="checkbox"/> Add <input checked="" type="checkbox"/> Edit <input checked="" type="checkbox"/> Delete				
Other Privileges <input type="text"/> +				
<input checked="" type="checkbox"/> Save Update				

User Entity Permission List

User Id	Role Id	Entity Id	Reference Id	View	Add	Edit	Delete	OtherPrivileges	Edit
		CaseDetail	39	True	False	False	False	-	E
		CaseDetail	4	True	True	True	True	Save	E

Show entries Showing 1 to 2 of 2 entries « « 1 » »

If we set the *Edit* permission for the *Case Detail* entity *Case Number 3* we the field level permission will set it and can check the privileges in the UI by using the following code as example.

```
column.ForColumn(col => CelloSaaS.ServiceProxies.AccessControlManagement.
    UserEntityPermissionProxy.CheckUserEntityPermissionByEntityOperation(
        col.EntityIdentifier,
        userIdentifier,
        col.Identifier,
        CelloSaaS.Model.EntityOperation.Update,
        tenantIdentifier) ?
    "<a onclick=\"javascript:return addCaseTab(this);\" href='"
    + Url.Action("EditCase", new {
        caseId = col.Identifier,
        tenantId = col.TenantId })
    + "' title=\"" + this.GetGlobalResourceObject("CustomerCaseResou
rce"

        , "EditCase").ToString()
    + " '" + col.CaseNumber + "'>" + strEditImageUrl
    + "</a>" : "-")
    .Named(this.GetGlobalResourceObject("General", "Edit").ToString())
    .Attributes(@class => "halign")
    .DoNotEncode());
```

The result will be shows like this if the *Edit* permission is **true**



The screenshot shows a 'Case Details' page with a header containing a magnifying glass icon and the text 'Case Details'. Below the header is a navigation bar with two tabs: 'Open Cases' (selected) and 'Closed Cases'. The main content area contains a table with the following data:

Case Type	Case Status	Case Sub Status	Case Id	Case Source	Edit
Sales	Open	None	3	Phone	

If we set **false** to the *Edit* permission the *Edit* permission will not be available for the user.



The screenshot shows a 'Case Details' page with a header containing a magnifying glass icon and the text 'Case Details'. Below the header is a navigation bar with two tabs: 'Open Cases' (selected) and 'Closed Cases'. The main content area contains a table with the following data:

Case Type	Case Status	Case Sub Status	Case Id	Case Source	Edit
Sales	Open	None	3	Phone	-

How To Guide: [Security Management](#)

8.8 Audit Trail

For better security, it is mandatory in the SaaS product to capture the audit information of important entities. For example, in a CRM system all the changes made on a customer entity needs to be tracked. CelloSaaS allows the developers to provide this feature with less effort.

Usage: The entities should inherit from CelloSaaS.Model.BusinessEntities. It then derives the following properties which are used to track:

IsTrackingOn

TrackChanges

GetChanges

ApplyChanges

IsTrackingOn indicates that the business object need to be trackable or not. TrackChanges will set IsTrackingOn and get the clone of the current object by using CloneUtiliy which is provided by CelloSaaS and store cloned object in baseline of change object.
`changes.Baseline = CloneUtility.DeepClone(obj);`

GetChanges will copy the changed object, build the change by using ChangeBuilder provided by CelloSaaS and returns the change object.

`ChangeBuilder objectDifferenceUtility = newChangeBuilder(changes);objectDifferenceUtility.BuildChange();`

ApplyChanges will reset IsTrackingOn and baseline of the change object.

How can we track our business object?

Configuration: To bring the tracking feature, we have to do the following configuration in web.config file.

Add the tracking provider section in configsections.

```
<sectionname="TrackingProvider" type ="CelloSaaS.Library.Configuration.ProviderConfiguration,
CelloSaaS.Library"/>
<TrackingProviderDefaultProvider="DefaultProvider">
<Providers>
<addname="DefaultProvider" assembly="CelloSaaS.Library" type="CelloSaaS.Library.Tracking.TrackingAdapter"/>
</Providers>
</TrackingProvider>
```

If you want to track your changes in different database, CelloSaaS provides you to configure the tracking connection string.

```
<addname="TrackingConnectionString" connectionString="Data Source=ServerName;Initial
Catalog=DbName;User Id=UserId;Password=password;" providerName ="System.Data.SqlClient"/>
```

This is optional. If you do not give tracking connection string, by default CelloSaaS will take the application connection string.

Database: From Database side, you will have to create history table based on the following structure in which tracking changes will be stored. Table Name must be actual schema table name with suffix "**History**" ie. For Example: Consider Employee table, the history table should be "**EmployeeHistory**".

Column Name	Data Type	Allow Nulls
-------------	-----------	-------------

HistoryTableNameID	Varchar (50)	
ReferenceID	Varchar (50)	
Snapshot	XML	
Changes	XML	
ChangedBy	Varchar (50)	
ChangeAction	Varchar (50)	
ChangeDate	Datetime	

Coding

To track the inserting value, you will have to set IsTrackingOn before calling Insert operation.

```
employeeDetails.IsTrackingOn = true;
employeeDetailsDAL.Create(createRequest);
```

To track the updating value, you will have to set tracking after fetching the baseline object.

```
employeeDetails = employeeDetailsDAL.Fetch(dataFeatchRequest);
employeeDetails.TrackChanges();
```

To track the deleting value, you will have to set tracking after fetching the baseline object.

```
employeeDetails = employeeDetailsDAL.Fetch(dataFeatchRequest);
employeeDetails.TrackChanges();
```

To fetch the changes for your business model object, you will have to call GetChanges provided by CelloSaaS in tracking service.

```
ITrackingService trackingService = newTrackingService();
Changes change = trackingService.GetChanges(entityName, entityReference, startDate, endDate);
```

EntityName: Entity Identifier of business object

EntityReference: Record Id of business object

StartDate,EndDate: Used to fetch the changes for given period of date

Change object will have the following properties:

BaseLine: Copy of the business object when object gets started to track

Current: Copy of the business object after object gets changed

Delta: Copy of the business object which has only differences between baseline and current

ChangeSet: Set of property identifiers with changed status

ChangedDate: Tracking date

ChangedBy: Logged in user Id

8.9 Triple DES Password Encryption Service

The implementation of CelloSaaS Password encryption is changed to Triple DES, the common name for the Triple Data Encryption Algorithm (TDEA) block cipher. Because of the availability of increasing computational power, the key size of the original DES cipher was becoming subject to brute force attacks. Triple DES is designed to provide a relatively simple method of increasing the key size of DES to protect against such attacks, without designing a completely new block cipher algorithm.

CelloSaaS allows to set tenant based CryptKey by using DB store provider.

8.9.1 Implementation of Triple DES

For those who are trying to migrate from CelloSaaS ver 2.1 to ver 2.2, use the PasswordCrypt Utility (Available in Package) to update the password with new encryption mechanism [This will Add reference to CelloSaaS.EncryptionProvider.dll].

Add the below config in the typeAlias Section of Unity.config

```
peAliasalias="ITenantCryptKeyDAL" type="CelloSaaS.EncryptionProvider.DAL.ITenantCryptKeyDAL
, CelloSaaS.EncryptionProvider" />
<typeAliasalias="TenantCryptKeyDAL" type="CelloSaaS.EncryptionProvider.SqlDAL.TenantCryptKe
yDAL, CelloSaaS.EncryptionProvider" />
    <!-- Uncomment the following line for MySql Database & comment the above line -->
<!--<typeAlias alias="TenantCryptKeyDAL"
type="CelloSaaS.EncryptionProvider.MySqlDAL.TenantCryptKeyDAL,
CelloSaaS.EncryptionProvider" />-->
<typeAliasalias="ITenantKeyStoreService" type="CelloSaaS.Library.Encryption.ITenantKeyStoreSer
vice, CelloSaaS.Library" />
<typeAliasalias="DBStore" type="CelloSaaS.EncryptionProvider.DBStore,
CelloSaaS.EncryptionProvider" />
<typeAliasalias="IPasswordEncryptionService" type="CelloSaaS.Library.Encryption.IPasswordEncrptionService, CelloSaaS.Library" />
<typeAliasalias="TripleDESPasswordEncryptionService" type="CelloSaaS.EncryptionProvider.Triple
DESPasswordEncryptionService,
```

Add the following in DataAccess Container:

```
<typetype="ITenantCryptKeyDAL" mapTo="TenantCryptKeyDAL" name="ITenantCryptKeyDAL" />
<typetype="ITenantKeyStoreService" mapTo="DBStore" name="ITenantKeyStoreService" />
<typetype="IPasswordEncryptionService" mapTo="TripleDESPasswordEncryptionService" name="IPa
sswordEncryptionService"
```

8.10 CelloSaaS - Single Sign On

CelloSaaS supports federated authentication i.e single sign on. By using this feature, you can connect your existing LDAP to authenticate the users. This feature is tenant wise configurable, so for each tenant the user will be redirected to their Identity Provider according to the settings.

Prerequisites

Windows 7 or Windows Vista or Windows Server 2003 or Windows Server 2008

Windows Identity Foundation SDK. <http://www.microsoft.com/download/en/details.aspx?id=4451>

IIS 7.0

ADFS 2.0 – For installing and configuring ADFS 2.0 server please refer the MSDN Link.

[http://technet.microsoft.com/en-us/library/adfs2-step-by-step-guides\(WS.10\).aspx](http://technet.microsoft.com/en-us/library/adfs2-step-by-step-guides(WS.10).aspx)



ADFS 2.0 server and Application Server should be on separate machine.

SSO Setup Overview

CelloSaaS Web application also known as Relying Party Site (RP Site) at

<https://www.cellosaassite.com>

CelloSaaS IdP also known as Custom IdP Site (Custom-IdP Site) at <https://login.cellosaas.com>

ADFS 2.0 Server (RP-IdP) at <https://adfs2.company.com>



RP Site will only trust RP-IdP and accepts its claims. RP-IdP should be configured to trust Custom-IdP Site.

For each tenant RP-IdP must be configured to trust the tenant's ADFS-2.0.

All sites must have HTTPS binding and valid certificates. Each site must have its own unique certificate.

Self signed certificates can be used for development environment purposes. Please refer "Creating Self Signed Certificates for Development Use" section for creating self signed certificates.

CelloSaaS Custom Identity Provider

If a particular tenant does not have their LDAP server then normal CelloSaaS authentication can be used. For such tenants they will be redirected to this Identity Provider Site. This site provides the usual Log-in screen. CelloSaaS-IdP Site will be found in the Release Package (v3.1+). Change the connection string and host this site in IIS7.0. You must enable HTTPS support for this site.

Change the following in the web.config:

```
<addkey="IssuerName" value="CertificateIssuerName"/>
<addkey="SigningCertificateName" value="CN=login.cellosaas.com"/>
<addkey="EncryptingCertificateName" value="" />
<addkey="EnableAppliesToValidation" value="true"/>
<addkey="RPAddress" value="https://www.cellosaassite.com"/>
```

You should have valid certificates to use them here (Can be self signed for development).

Changes required to enable SSO in CelloSaaS Web Application:

You need to modify the web.config as follows to enable the SSO. (For convenience, we have provided Web-SSO.config file which contains complete configuration for enabling SSO).

In <configSections> add

```
<sectionname="microsoft.identityModel" type="Microsoft.IdentityModel.Configuration.MicrosoftIdentityModelSection, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
```

In <appSettings> add

```
<addkey="EnableFederation" value="true" />
<addkey="DefaultHomeRealm" value="default home realm" />
```

In <IdentityProvider> add

```
<addname="ClaimsIP" assembly="CelloSaaS.Library" type="CelloSaaS.Library.ClaimUserIdentityProvider" /> inside the <providers> element
Change the defaultProvider="Default" to defaultProvider="ClaimsIP"
```

In the <assemblies> elements add

```
<addassembly="Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" />
<addassembly="System.Design, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=B03F5F7F11D50A3A" />
```

In <system.web><httpModules> and <system.webServer><modules>

```
<addname="WSFederationAuthenticationModule" type="Microsoft.IdentityModel.Web.WSFederation
AuthenticationModule, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
<addname="SessionAuthenticationModule" type="Microsoft.IdentityModel.Web.SessionAuthenticatio
nModule, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
```

Add this before </configuration> element

```
<microsoft.identityModel>
<service>
<audienceUris>
<addvalue="https://www.cellosaassite.com" />
</audienceUris>
<serviceCertificate>
<certificateReference>x509FindType="FindByThumbprint"findValue="thumb print please
change"storeLocation="LocalMachine"storeName="My" />
</serviceCertificate>
<applicationService>
<claimTypeRequired>
<claimType>type="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"optional="true" />
<claimType>type="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"optional="tru
e" />
</claimTypeRequired>
</applicationService>
<certificateValidation>certificateValidationMode="None" />
<federatedAuthentication>
<wsFederationpassiveRedirectEnabled="false"
issuer="https://adfs2.company.com/adfs/ls/"
realm="https://www.cellosaassite.com/"
reply="https://www.cellosaassite.com/Account/FederatedResult/"requireHttps="true" />
<cookieHandlerrequireSsl="true" />
</federatedAuthentication>
<issuerNameRegistry>type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry
, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">
<trustedIssuers>
<addthumbprint="paste the adfs server thumbprint
here">name="http://adfs2.company.com/adfs/services/trust" />
</trustedIssuers>
</issuerNameRegistry>
</service>
</microsoft.identityModel>
```

Add this after </system.webServer> element

```
<locationpath="FederationMetadata">
<system.web>
<authorization>
<allowusers="*" />
</authorization>
```

```
</system.web>
</location>
<locationpath="Account/CompanyCode">
<system.web>
<authorization>
<allowusers="*" />
</authorization>
</system.web>
</location>
```



To test the SSO feature, you must host your application in the IIS 7.0 with HTTPS binding.
Visual Studio built in web developer server cannot be used for SSL.

Update the Account Controller with the following code:

```
public ActionResult LogOn(string companyCode)
{
    if (System.Threading.Thread.CurrentPrincipal.Identity.IsAuthenticated)
    {
        // user already loged in so redirect him/her to home page
        return RedirectToAction("Index", "Home");
    }
    if (!ConfigHelper.EnableFederation)
    {
        return View(); //proceed to normal sign in
    }
    if (!string.IsNullOrEmpty(Request.QueryString["werror"]))
    {
        return FederatedSignOff();
    }
    else
    {
        string tenantId = string.Empty;
        string whr = ConfigHelper.DefaultHomeRealm;

        if (string.IsNullOrEmpty(companyCode))
        {
            string url = Request.Url.AbsoluteUri.Substring(0, Request.Url.AbsoluteUri.Length - Request.Url.PathAndQuery.Length);

            TenantDetails tenantDetails = TenantProxy.GetTenantDetailsByURL(url);

            if (tenantDetails != null && !string.IsNullOrEmpty(tenantDetails.TenantCode))
            {
                tenantId = tenantDetails.TenantCode;
            }
        }
        else
        {
            // if tenant details not found return a page to enter the company code
            return View("CompanyCode");
        }
    }
}
```

```

// get tenantid from the entered company code
tenantId = TenantProxy.GetTenantIdFromTenantCode(companyCode);

if (string.IsNullOrEmpty(tenantId))
{
    ModelState.AddModelError("", "Company Code Not Found!");
    return View("CompanyCode");
}

whr = GetHomeRealmForTenant(tenantId) ?? whr;

var fam = FederatedAuthentication.WSFederationAuthenticationModule;

SignInRequestMessage mess = new SignInRequestMessage(
newUri(fam.Issuer),
    fam.Realm, fam.Reply ?? fam.Realm)
{
    Context = tenantId,
    HomeRealm = whr
};
Response.Redirect(mess.WriteQueryString());
}

return View();
}

[ValidateInput(false)]
public ActionResult FederatedResult()
{
    var fam = FederatedAuthentication.WSFederationAuthenticationModule;

    if (fam.CanReadSignInResponse(System.Web.HttpContext.Current.Request, true))
    {
        // This is a response from the STS
        SignInResponseMessage mesg = WSFederationMessage.CreateFromNameValueCollection(
WSFederationMessage.GetBaseUrl(Request.Url), Request.Form) as SignInResponseMessage;

        string userName = GetUserNameFromClaims(System.Threading.Thread.CurrentPrincipal);
        string tenantId = mesg.Context;
        string companyCode = string.Empty;
        string tokenXML = mesg.Result;

        Tenant tenantInfo = TenantProxy.GetTenantInfo(tenantId);
        companyCode = tenantInfo.TenantDetails.TenantCodeString;

        //Get user details based on the user name
        UserDetails userDetails = UserDetailsProxy.GetUserDetailsByName(userName, tenantId);

        if (userDetails != null && userDetails.MembershipDetails != null
&& !string.IsNullOrEmpty(userDetails.MembershipDetails.TenantCode))
        {
            // audit the login
            string eventId = newEventSchedulerService().GetEventByName("Login");

            if (!string.IsNullOrEmpty(eventId))
            {
                ContextHelper.SetOperatorTenantContext(userDetails.MembershipDetails.TenantCode);
            }
        }
    }
}

```

```

EventRegister eventRegister = newEventRegister();
ContextHelper.SetOperatorTenantContext(tenantId);
    eventRegister.RegisterEvent(null, newEvent
    {
        EventId = eventId,
        EventDescription = "User Login",
        UserId = userDetails.User.UserId
    });

ContextHelper.ClearOperatorTenantContext();
}

// fetch tenant roles, privileges and license and set form auth ticket cookies
// add necessary claims to the principal
if (this.PrepareTenantRolesAndPrivileges(userDetails))
{
    // redirect to dashboard
    return RedirectToAction("Index", "Home");
}
else
{
    ModelState.AddModelError("Login", "User Details are not available!");
}
else
{
    ModelState.AddModelError("Request", "Not a valid Federation SignIn Response Request!");
}

FederatedAuthentication.SessionAuthenticationModule.SignOut();
FederatedAuthentication.SessionAuthenticationModule.DeleteSessionTokenCookie();
FederatedAuthentication.WSFederationAuthenticationModule.SignOut(false);
FormsAuthentication.SignOut();

return View("FederatedSignInError");
}

private static string GetUserNameFromClaims(System.Security.Principal.IPrincipal iPrincipal)
{
    var claims = ((IClaimsPrincipal)System.Threading.Thread.CurrentPrincipal).Identities[0].Claims;
    foreach (var item in claims)
    {
        if (item.ClaimType == ClaimTypes.Name)
        {
            return item.Value;
        }
    }
    return string.Empty;
}

private static string GetHomeRealmForTenant(string tenantId)
{
    string whr = null;
    var tenantSettings = TenantSettingsProxy.GetTenantSettings(tenantId);
    if (tenantSettings != null && tenantSettings.Settings != null
        && tenantSettings.Settings.Attributes != null)

```

```

&& tenantSettings.Settings.Attributes.ContainsKey(AttributeConstants.HomeRealm)
&&!string.IsNullOrEmpty(tenantSettings.Settings.Attributes[AttributeConstants.HomeRealm]))
{
    whr = tenantSettings.Settings.Attributes[AttributeConstants.HomeRealm];
}
return whr;
}

publicActionResult LogOff()
{
if (Thread.CurrentPrincipal is IClaimsPrincipal)
{
    return FederatedSignOff();
}
else
{
    FormsAuthentication.SignOut();
}
return RedirectToAction("LogOn", "Account");
}

privateRedirectResult FederatedSignOff()
{
    FederatedAuthentication.SessionAuthenticationModule.SignOut();
    FederatedAuthentication.SessionAuthenticationModule.DeleteSessionTokenCookie();
    FederatedAuthentication.WSFederationAuthenticationModule.SignOut(false);
    Response.Cache.SetCacheability(HttpCacheability.NoCache);
    WSFederationAuthenticationModule fam =
    FederatedAuthentication.WSFederationAuthenticationModule;
    SignOutRequestMessage signOutRequest = new SignOutRequestMessage(new Uri(fam.Issuer),
    fam.Realm);
    return Redirect(signOutRequest.WriteQueryString());
}

```

Creating Self Signed Certificates for Development Use:
Enter the following commands in the Visual Studio Command Prompt

 **Step 1:** Create a Certificate Authority

```

makecert -n "CN=Company Certificate Authority"
    -cy authority
    -a sha1
    -sv "CompanyPrivateKey.pvk"
    -r
    "CompanyCA.cer"

```

This command will prompt for password. Please provide it.
This will create CompanyCA.cer file.

 **Step 2:** Now create certificate for your test site

```

makecert -n "CN=www.cellosaassite.com"
    -ic "CompanyCA.cer"
    -iv "CompanyPrivateKey.pvk"
    -a sha1
    -sky exchange
    -pe
    -sv "www.cellosaassite.comPrivateKey.pvk"
    "www.cellosaassite.com.cer"

```



This will create www.cellosaassite.com.cer file.

Step 3: Create a spc file

```
cert2spc www.cellosaassite.com.cer www.cellosaassite.com.spc
```

```
cert2spc CompanyCA.cer CompanyCA.spc
```

This will create www.cellosaassite.com.spc and CompanyCA.spc files.

Step 4: Now combine them into pfx file and import them in mmc

```
pvk2pfx -pvk "www.cellosaassite.comPrivateKey.pvk"
```

```
    -spc "www.cellosaassite.com.spc"
```

```
    -pfx "www.cellosaassite.com.pfx"
```

```
    -pi YourPassword
```

```
pvk2pfx -pvk "CompanyPrivateKey.pvk"
```

```
    -spc "CompanyPrivateKey.spc"
```

```
    -pfx "CompanyPrivateKey.pfx"
```

```
    -pi YourPassword
```

This will create www.cellosaassite.com.pfx and CompanyPrivateKey.pfx

Import www.cellosaassite.com.pfx in MMC, the MY store of localmachine

Import CompanyPrivateKey.pfx to the root store of local machine in MMC



The domain name of the site, to use this certificate should be www.cellosaassite.com, else it will always throw an error.

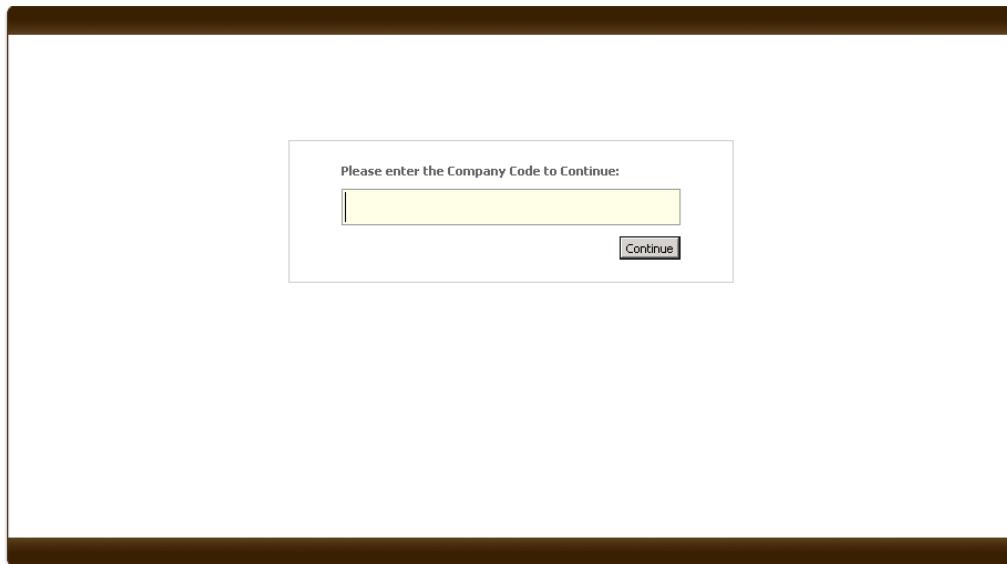
Working

If the tenant is identified by the requesting URL, and if the tenant setting has the Home Realm parameter set, then the user is automatically redirected to that Identity provider site.

If the tenant does not set the home realm parameter, then the user is redirected to the default home realm that is set in the web.config file with redirects to the custom celloaas identity provider.

Welcome to techcello

World's Most Innovative SaaS Framework



The image shows a screenshot of a web browser window. At the top, there is a navigation bar with several icons. Below the navigation bar, the main content area displays a login form. The form has a light gray background and contains the following text and input fields:
Please enter the Company Code to Continue:

Figure 8-11 – Techcello Login Screen

If the tenant cannot be identified by the URL, then the application pops-up for the company code to continue for the login.

8.11 Admin Data Management

MasterData

Most of the systems possess lot of administration screens and building them consumes time. Master data in CelloSaaS facilitates the creation of admin screens with just few lines of code in short span of time. Master data is used in building a data entry page for a table by simply mentioning the configuration details in config file. The data entered through this page will be attached to the specific tenant.

MasterData Configuration

To configure master data, the user has to specify the details in a separate configuration file. *MasterDataConfiguration.Config*.

The user needs to specify the configurable settings using appropriate tags and attributes as follows:

```
<masterData>
<addname="Language"entityName="Language"tableName="Language"viewPrivilege="ViewMaster
Data"createPrivilege="addMasterData"deactivateRecord="true">
<auditInformationname="Language"tenantColumn="TenantId"statusColumn="LanguageStatus"prim
aryColumn="Language_ID">
</auditInformation>
<column>
<addname="Language_ID"fieldId="Language_ID"displayHeader="Language
ID"parentValueColumn=""displayAs="AsIIts"requiredErrorMessage="Required"mandatory="True"></
add>
    <addname="Language_Name"fieldId="Language_Name"displayHeader="Language
Name"displayAs="AsIIts"regularExpression="[^a-zA-Z]"mandatory="True"></add>
<addname="Hi"fieldId="Hi"displayHeader="Hi"parentValueColumn=""displayAs="AsIIts"parentIdCol
umn=""parentTable=""mandatory="false"></add>
</column>
<childRelations>
    <addname="Area"childTable="Area"childForeignKeyColumnName="LanguageId"childTable
SoftDelete="true"></add>
</childRelations>
</add>
<addname="Area"entityName="Area"tableName="Area">
<auditInformationname="Area"tenantColumn="TenantId"primaryColumn="AreaId"statusColumn="Ar
eaStatus"></auditInformation>
<column>
<addname="AreaName"fieldId="AreaName"displayHeader="Area Name"displayAs="AsIIts"></add>
<addname="AreaDescription"fieldId="AreaDescription"displayHeader="Area
Description"displayAs="AsIIts"></add>
<addname="LanguageId"fieldId="LanguageId"displayHeader="Language"displayAs="DDL"parentId
Column="Language_ID"parentTable="Language"parentValueColumn="Language_Name"parentTen
antColumn="TenantId"parentStatusColumn="LanguageStatus"></add>
</column>
</add>
<addname="Department"entityName="Department"tableName="Department">
<auditInformationname="Department"tenantColumn="TenantId"primaryColumn="DepartmentId"></a
uditInformation>
<column>
<addname="DepartmentName"fieldId="DepartmentName"displayHeader="Department
Name"displayAs="AsIIts"></add>
<addname="DepartmentDesc"fieldId="DepartmentDesc"displayHeader="Department
Desc"displayAs="AsIIts"maxLength="100"></add>
</column>
<childRelations>
<addname="Designation"childTable="Designation"childForeignKeyColumnName="DepartmentId"><
```

```
/add>
</childRelations>
</add>
</masterData>
```

MasterDataConfiguration Tags`

In order to configure master data, user should know the list of tags used. <masterData> tag – MasterDataElement: This tag is used to configure the masterdata. The attributes of this tag should be used with the <add> tag. The following are the attributes related to this tag:

Attributes	Description
Column	Used to get the columns of the entity. Value of this attribute should be MasterDataColumnElement name
EntityName	Specifies the entityname which is used to bring out CelloSaaS features like Data model extension,tracking, etc.
TableName	Specifies the name of the table
ConnectionStringName	Specifies the name of the connection string. If it does not exist, CelloSaaS considers Application connection string
CreatePrivilege	Sets create privilege to add a masterData
UpdatePrivilege	Sets update privilege to update the masterdata
DeletePrivilege	Sets delete privilege to delete the masterdata
ViewPrivilege	Sets view privilege to view the masterdata list and details

The following elements help in configuring the complete masterdata based on the user requirements. All these elements should be configured under <masterData> tag.

1.<auditInformation> tag – AuditFieldInformation:This tag is used to audit masterdata entity. The following are the attributes related to this tag.

Attributes	Description
StatusColumn	Attribute is used to identify the audit status of the column. The value for this attribute is any valid string. For example, statusField="Status"
TenantColumn	Specifies name of the column where tenantId will be stored
primaryColumn	Specifies name of the primary column
UpdatedBy	Specifies the user who had lastly updated the masterdata entity row. The value for this attribute is the username who had lastly updated the row

UpdatedOn	Specifies the date in which the masterdata entity row lastly updated. The value for this attribute is the date string specifying when the row got updated
AddedBy	Specifies the user who added the masterdata entity row. The value for this attribute is the username who had added the row
AddedOn	Specifies the date in which the masterdata entity row got added. The value for this attribute is the date string specifying when the row got added

2.<column> tag - MasterDataColumnElement: This tag is used to define the list of columns that should get bound with the gridview. The attributes of this tag should be used with the <add> tag. Following are the attributes related to this tag.

Attributes	Description
Name	Attribute used to uniquely identify a column from other columns. The value for this attribute is any valid string.
FieldId	Specifies the field Identifier which is used to bring out the customization
Display Header	Specifies the column header name. The value for this attribute is any valid string.
Display Text Column	Specifies what text to be displayed to the user for the column. The value for this attribute is any valid string.
Display As	Specifies how the column should get displayed to the user. The value for this attribute can be any one of the following: AsTxt – To display a textbox control DDL – To display a drop down list control CheckBox – To display a checkbox control Radio – To display a Radio button control Date – To display a date control PickupList – To display a pickup list which has been added in pickup list configuration
Parent Table	Used to specify whether the column got any key relation with some other table. The value for this attribute is any valid database table name which got a key relation with this column
Parent value Column	Specifies to which parent table column the field got the key relation. The value for this attribute is the column name which got the key relation with this column
Parent Tenant column	Specifies name of the parent column where tenantId will be stored
Parent Status column	Specifies name of the parent status column
Mandatory	It specifies whether user have to provide input for the column or can be left empty True: specifies that user have to provide input for the column False: specifies the column can be left empty
MultiLine	Specifies whether the text field is multiline text field or singleline text field True: specifies that the column is a multiline text field False: specifies that the column is a singleline text field
MaxLength	Specifies the max length for text box
Required Error	Specifies the error message that should be displayed to the user

Message	when the required/mandatory column left blank The value for this attribute is any valid string
Read Only	Specifies whether the column can be modified by the user or not True: specifies that the column cannot be modified by the user False: specifies that the column can be modified by the user
AutoPostBack	Specifies whether the column should posts back to the server each time a user interacts with the column or not True: specifies that the column should post back to the server False: specifies the column should not post back to the server
Soft Delete	Specifies whether to introduce a flag corresponding to the column which indicates whether the column should be logically deleted or should be actually deleted without any introduction of a flag True: specifies that a flag should be introduced in order to do a logical delete False: specifies the column should be deleted actually
PickupListId	Specifies Pickuplist id which has been created in pickuplist management

3.<childRelations> tag: This tag is used to configure the child relation of master data. Usually used to represent a hierarchical gridview. The attributes of this tag should be used with the <add> tag. The following are the attributes related to this tag:

Attributes	Description
Child Table	Specifies the child table that got a key relation with this column. The value for this attribute is any valid database table name which got a key relation with this column
Child Foreign Key Column Name	Specifies the child column name to which this column got a key relation. The value for this attribute is the column name which got the key relation with this column
Child Table Soft Delete	Specifies whether to introduce a flag corresponding to the child column which indicates whether the child column should be logically deleted or should be actually deleted without any introduction of a flag True: specifies that a flag should be introduced in order to do a logical delete False: specifies that the child column should be deleted actually Based on the test result, the data limit for the tables of master data can be in the range of 500-800 records
ChildTenantColumn	Specifies tenant column for the child table
ChildStatusColumn	Specifies status column for the child table

9 SUBSCRIPTION

Unlike traditional software which is conventionally sold as a perpetual license with an up-front cost and an optional ongoing support fee, SaaS Application are generally priced based on subscription fee, most commonly a monthly fee or an annual fee [Flexible Billing Cycle]. Consequently, the initial setup cost for SaaS is typically lower than the equivalent enterprise software or Zero. SaaS vendors typically price their applications based on some usage parameters, such as the number of users ("seats") using the application, Number of Features/Modules they have subscribed to etc. However, because in a SaaS environment customers' data resides with the SaaS vendor, opportunities also exist to charge per transaction, event, or other unit of value.

The relatively low cost for **user provisioning** (i.e., setting up a new Tenant/customer) in a multi-tenant environment enables some SaaS vendors to offer applications using the **freemium**/Demo model. In this model, a free service is made available with limited functionality or scope, and fees are charged for enhanced functionality or larger scope.

A key driver of SaaS growth is SaaS vendors' ability to provide a price that is competitive with on-premises software. CelloSaaS Provides a Package management module, this provides the ability to group one or more features/components of the System and make the customers to choose any one of the pre-set package or build a custom package exclusively for this customer. This gives the ability to the vendor to be more flexible with the solutions that he provides and compete with an outside service provider may be able to offer better, cheaper, more reliable applications.

9.1 Manage Subscription

Every SaaS application would package the modules and features in various packages and have the tenants / end users subscribe for the usage. CelloSaaS comes along with subscription management capabilities.

Developers can extend these features to the product requirements.
Package is collection of Service, Modules, Features and Usage, decided by the ISV (SaaS application owner).

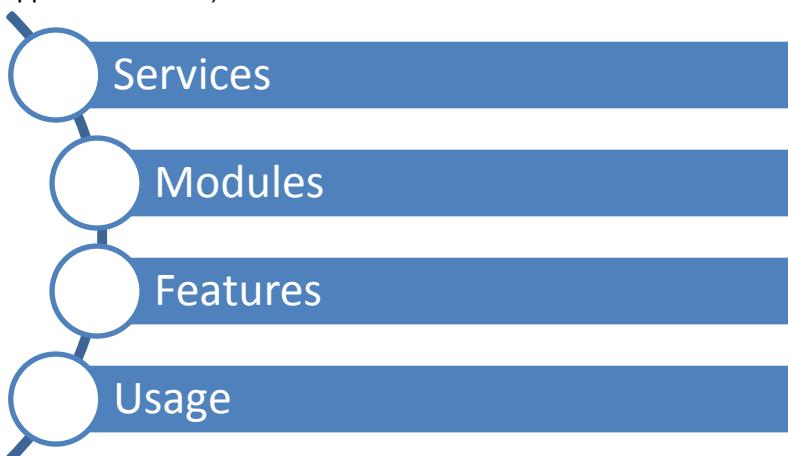


Figure 9-1 - Package Flow Diagram

Service is a logical grouping of set of modules. For example, "IDM is a service which consists of employee management modules. Module is a logical grouping of set of features. For example, "Employee Management" is a module which consists of features. Feature is a logical representation of any functionality. For example, "Employee Profile Management", "Organization Hierarchy Management" are features. Usage represents any metric based restrictions that we want in the system. For example, "Number of customers created" in a CRM application can be usage based restriction.

Each tenant will subscribe to a package, based on their need.

Module can exist without service mapping. Those modules will be available under “Other modules”.

Product admin will select the package for tenant based on what they intend to purchase.

Administration

Services

Each Service contains modules, features and usages.

Use Services table to add service details.

Services table details.



Services	
Service_Code	
Service_Name	
Service_Description	
Service_CreatedBy	
Service_CreatedOn	
Service_UpdatedBy	
Service_UpdatedOn	
Service_Status	

Figure 9-2 – Services Screen

Example

Service_Code	Service_Name	Service_CreatedBy	Service_CreatedOn	Service_Status
CelloSaaS_Service	CelloSaaS_Service	Admin	2009-09-10 11:59:01.723	1

```
To get all module details
///<summary>
/// Get all the service details
///</summary>
///<returns>Service details</returns>
public Dictionary<string, Service> GetAllServices();
```

Modules

CelloSaaS has the following modules by default:

- Tenant, User, Package, Settings, Access Control and Configuration.

Each module contains features and usages.

Use modules table to add module details.

Modules	
Module_Code	
Module_Name	
Module_ServiceCode	
Module_CreatedBy	
Module_CreatedOn	
Module_UpdatedBy	
Module_UpdatedOn	
Module_Status	

Figure 9-3 – Module Table Details Screen

Example

Module_Code	Moule_Name	Module_CreatedBy	Module_CreatedOn	Module_Status
User	User	Admin	2009-09-10 11:59:01.723	1

To get all module details

```
///<summary>
    /// Get all the module details
    ///</summary>
    ///<returns>Module details</returns>
    publicDictionary<string, Module> GetAllModules()
```

To get Module Details based on Service

```
///<summary>
    /// Fetching all the module details for each modules
    ///</summary>
    ///<param name="moduleId">Module Identifiers</param>
    ///<returns>feature details for each modules</returns>
    publicDictionary<string, Dictionary<string, Module>> GetModuleByServiceIds(string[] serviceIds)
```

Features

Features are module based and are stored in Feature table.

CelloSaaS has following features by default.

Feature	Module
ManageTenant	Tenant

ManageTenantLicense	Tenant
SelfTenantAdministrator	Tenant
ManagePackage	Package
ManageUser	User
ManageAllUser	User
ManageRole	AccessControl
ManageRoleFeatures	AcessControl
ManageRolePrivileges	AccessControl
ManageUserRoles	AccessControl
ManageAllRole	AccessControl
ManagePackageSettings	Setting
ManageRoleSettings	Setting
ManageTenantSettings	Setting
ManagePickupList	Configuration
ManageEntityExtn	Configuration
Manage DataViewExtn	Configuration

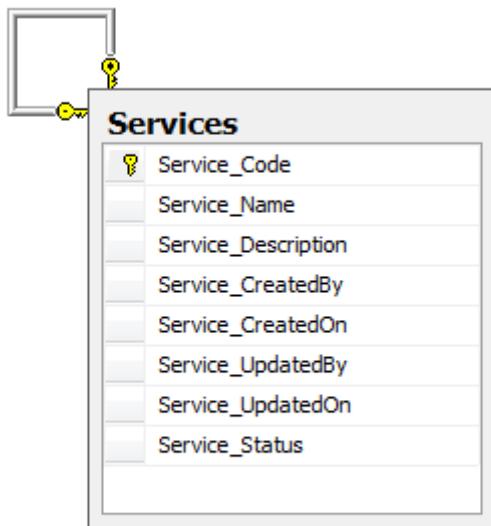


Figure 9-4 – Module Table Details Screen

Example

Feature_Code	Feature_Name	Feature_ModuleCode	Feature_CreatedBy	Feature_CreatedOn	Max Length
ManageUser	ManageUser	User	Admin	2009-09-10 11:59:08.000	1

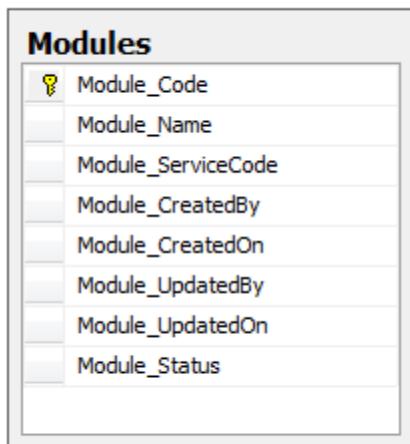


Figure 9-5 – Module Table Details Screen

To get feature details based on the module

///<summary>

/// Fetching all the feature details for each modules

///</summary>

///<param name="moduleId">Module Identifiers</param>

///<returns>feature details for each modules</returns>

publicDictionary<string, Dictionary<string, Feature>> GetFeatureByModuleIds(string[] moduleId)

Features are mapped with the privileges in FeaturePrivileges table

FeaturePrivilege table details

FeaturePrivileges	
FeaturePrivileges_Code	
FeaturePrivileges_FeatureCode	
FeaturePrivileges_PrivilegeC...	
FeaturePrivileges_CreatedOn	
FeaturePrivileges_CreatedBy	
FeaturePrivileges_UpdatedOn	
FeaturePrivileges_UpdatedBy	
FeaturePrivileges_Status	

Figure 9-6 – FeaturePrivileges Screen

Example

FeaturePrivileges_Code	FeaturePrivileges_FeatureCode	FeaturePrivileges_PrivilegeCode	FeaturePrivileges_CreatedBy	FeaturePrivileges_CreatedOn	FeaturePrivileges_Status
C49A5B8C-FE4F-4B8B-A2FD-258DD2E2EADB	ManageUser	Add_User	Admin	2009-09-10 11:59:01.000	1

Usages

Usages are module based

Usage details are stored in usage table

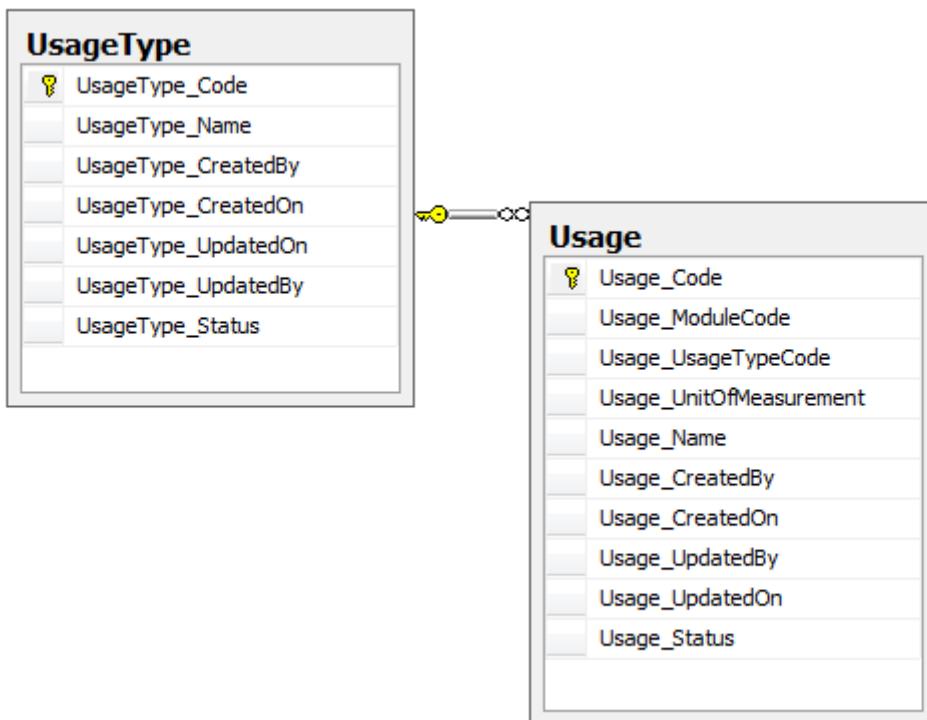


Figure 9-7 – Usage Table Detail Screen

Example

Usage_Code	Usage_ModuleCode	Usage_UsageTypeCode	Usage_UnitOfMeasurement	Usage_Name	Feature_CreatedBy	Feature_CreatedOn	Feature_Status
UserUsage1	User	841F2E69-056D-40FB-9652-8604360AB479	Numbers	UserUsage1	Admin	2009-09-10 11:59:08.000	1

To get feature details based on the module

```

///<summary>
    ///This method is used to get usage details for modules
///</summary>
///<param name="moduleId">ModuleIdentifiers</param>
///<returns>Usage Details for each modules</returns>
publicDictionary<string, Dictionary<string, Usage>> GetUsageByModuleIds(string[] moduleId)

```

Example

UsageType_Code	UsageType_Name	UsageType_CreatedBy	UsageType_CreatedOn	UsageType_Status
841F2E69-056D-40FB-9652-8604360AB49	BlockUsage	Admin	2009-09-10 11:59:08.000	1

License Package

Package details are stored in the LicensePackage table.

It contains package name and description.

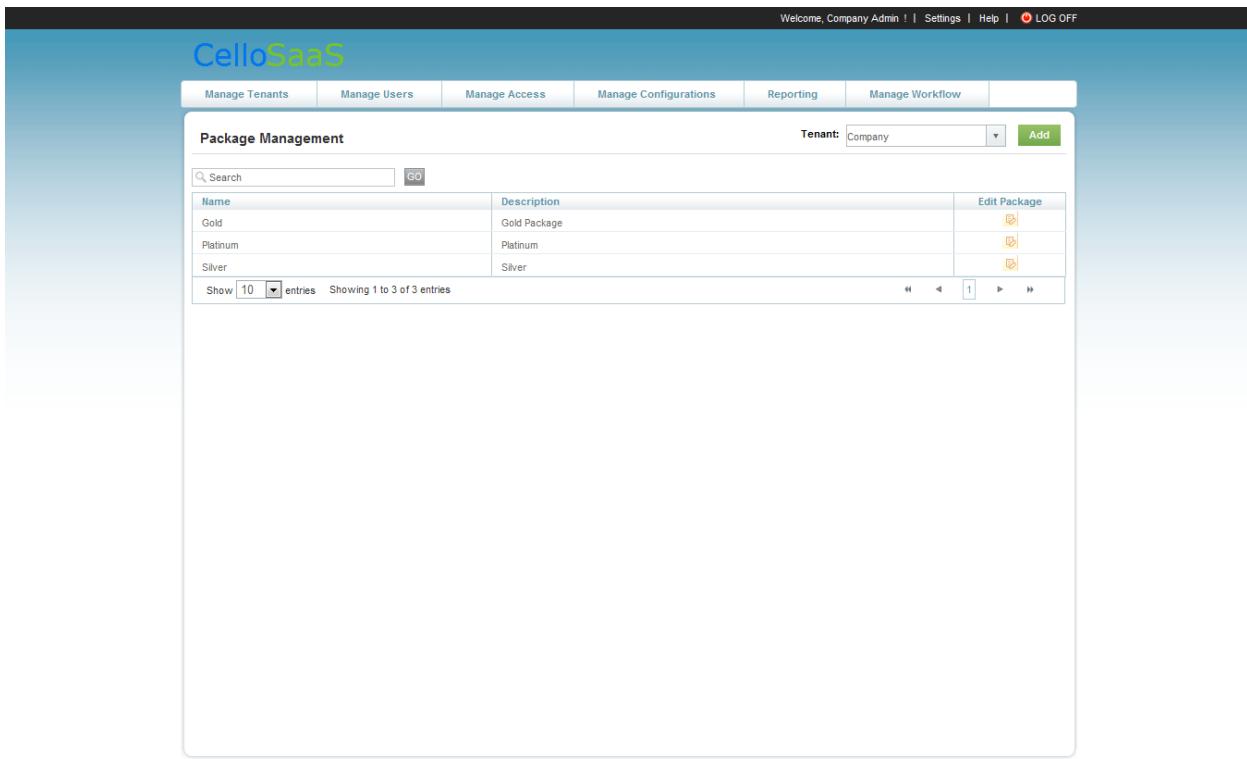
Package modules are stored in the LicensePackageModule table.

Package features and usages are stored in the LicensePackageFeature and LicensePackageUsage tables.

Product admin has permission to add/edit/delete package details.

Login as Product Admin, select Tenant Management -> Manage Package Details.

The page lists the active package details.



Name	Description	Edit Package
Gold	Gold Package	
Platinum	Platinum	
Silver	Silver	

Search GO

Show 10 entries Showing 1 to 3 of 3 entries

« < > »

Copyright © 2012 by techcello.com

Figure 9-8 – Package Management Screen

```
Get Active Packages Details
///<summary>
    /// This method is to get all the license package.
    ///</summary>
    ///<returns>List[LicensePackage]</returns>
public List<LicensePackage> GetAllLicensePackage()
```



```
Get Package Details by package Id
///<summary>
    /// Get the package details by packageid.
    ///</summary>
    ///<param name="packageId">Package Id</param>
    ///<returns>Package Details</returns>
public PackageDetails GetPackageDetailsByPackageId(string packageId)
```

Create Package

Click add button in package list page to create new package.

Give unique package name and its description.

Select modules from module details and the features will get populated corresponding to the module.

Select appropriate features for selected modules and click save button to create new package.

Welcome, Company Admin | Settings | Help | LOG OFF

CelloSaaS

Manage Tenants | Manage Users | Manage Access | Manage Configurations | Reporting | Manage Workflow

Manage Package Details

Tenant Name: Company

Package Details

Package Name: Gold | Package Description: Gold Package

Service Details

Services Not Available

Module Details

Assigned Modules	Assignable Modules
Other Modules	Other Modules
<input checked="" type="checkbox"/> Chart <input checked="" type="checkbox"/> Configuration <input checked="" type="checkbox"/> QueryBuilder <input checked="" type="checkbox"/> Tenant <input checked="" type="checkbox"/> Workflow <input checked="" type="checkbox"/> Setting <input checked="" type="checkbox"/> Package <input checked="" type="checkbox"/> AccessControl <input checked="" type="checkbox"/> User <input checked="" type="checkbox"/> SCA <input checked="" type="checkbox"/> Report	<input checked="" type="checkbox"/> Chart <input checked="" type="checkbox"/> Configuration <input checked="" type="checkbox"/> QueryBuilder <input checked="" type="checkbox"/> Tenant <input checked="" type="checkbox"/> Workflow <input checked="" type="checkbox"/> Setting <input checked="" type="checkbox"/> Package <input checked="" type="checkbox"/> AccessControl <input checked="" type="checkbox"/> User <input checked="" type="checkbox"/> SCA <input checked="" type="checkbox"/> Report

Feature Details

Assigned Features	Assignable Features
Chart	Chart
<input checked="" type="checkbox"/> ManageChart	<input checked="" type="checkbox"/> ManageChart
Configuration	Configuration
<input checked="" type="checkbox"/> Manage DataViewExtn <input checked="" type="checkbox"/> Manage Entity Extn <input checked="" type="checkbox"/> Manage Pickup List <input checked="" type="checkbox"/> Manage Setting Template <input checked="" type="checkbox"/> Manage Tenant Settings Template	<input checked="" type="checkbox"/> Manage DataViewExtn <input checked="" type="checkbox"/> Manage Entity Extn <input checked="" type="checkbox"/> Manage Pickup List <input checked="" type="checkbox"/> Manage Setting Template <input checked="" type="checkbox"/> Manage Tenant Settings Template
QueryBuilder	QueryBuilder
<input checked="" type="checkbox"/> ManageQuery	<input checked="" type="checkbox"/> ManageQuery
Tenant	Tenant
<input checked="" type="checkbox"/> Manage Tenant <input checked="" type="checkbox"/> Manage Tenant License	<input checked="" type="checkbox"/> Manage Tenant <input checked="" type="checkbox"/> Manage Tenant License
Workflow	Workflow
<input checked="" type="checkbox"/> Workflow Designer	<input checked="" type="checkbox"/> Workflow Designer
Setting	Setting
<input checked="" type="checkbox"/> Manage Package Settings <input checked="" type="checkbox"/> Manage Role Settings <input checked="" type="checkbox"/> Manage Tenant Settings	<input checked="" type="checkbox"/> Manage Package Settings <input checked="" type="checkbox"/> Manage Role Settings <input checked="" type="checkbox"/> Manage Tenant Settings
Package	Package
<input checked="" type="checkbox"/> Manage Package	<input checked="" type="checkbox"/> Manage Package
AccessControl	AccessControl
<input checked="" type="checkbox"/> Manage All Role <input checked="" type="checkbox"/> Manage Role <input checked="" type="checkbox"/> Manage Role Features <input checked="" type="checkbox"/> Manage Role Privileges <input checked="" type="checkbox"/> Manage User Roles <input checked="" type="checkbox"/> Tenant Access Feature	<input checked="" type="checkbox"/> Manage All Role <input checked="" type="checkbox"/> Manage Role <input checked="" type="checkbox"/> Manage Role Features <input checked="" type="checkbox"/> Manage Role Privileges <input checked="" type="checkbox"/> Manage User Roles <input checked="" type="checkbox"/> Tenant Access Feature
User	User
<input checked="" type="checkbox"/> Manage All User <input checked="" type="checkbox"/> Manage User	<input checked="" type="checkbox"/> Manage All User <input checked="" type="checkbox"/> Manage User
SCA	SCA
<input checked="" type="checkbox"/> Profile Feature	<input checked="" type="checkbox"/> Profile Feature
Report	Report
<input checked="" type="checkbox"/> Manage Report	<input checked="" type="checkbox"/> Manage Report

Cancel | Save

Figure 9-9 – Manage Package Detail Screen

```

Update package details
///<summary>
    /// Update the package details.
    ///</summary>
    ///<param name="licensePackageDetails">Package Details</param>
publicstring UpdatePackage(PackageDetails licensePackageDetails)

```

9.2 Tenant Subscription

Product admin will assign the appropriate package for the tenant during tenant creation. Tenant can have only one package.

Available active packages will be displayed in the license information of Tenant creation / Tenant update page. Product admin will assign package to tenant.

```

To add tenant license details
///<summary>
    /// This method is to insert the license for tenant.
    ///</summary>
///<param name="tenantLicense">Insert Tenant License Details</param>
    ///<returns>TenatLicenseld</returns>
publicstring InsertTenantLicense(TenantLicense tenantLicense)

```

```

Update tenant license
///<summary>
    /// This method is to update the license details of tenant.
    ///</summary>
///<param name="tenantLicense">Tenant License Details</param>
///<returns>TenantLicenseld</returns>
publicstring UpdateTenantLicense(TenantLicense tenantLicense)

```

```

Change tenant package / Update license
///<summary>
    /// This method is to update the tenant package details(Change Package).
    ///</summary>
///<param name="tenantLicense">Tenant license details</param>
publicvoid UpdateLicenseForLicensePackage(TenantLicense tenantLicense)

```

```

Get tenant license details
///<summary>
    /// This method is to get the tenant license modules, usages,features,privileges.
    ///</summary>
///<param name="tenantId">Tenant Identifier</param>
    ///<returns>Tenant License Details</returns>
publicTenantLicense GetTenantLicense(string tenantId)

```

9.3 Validate Subscription

Validate subscription is automatically executed on calling CelloSaaS login page. When you want to validate explicitly without using CelloSaaS Validation Page/Controller, call the appropriate Validation methods of CelloSaaS to check the Subscription Validation.

```

To validate subscription
///<summary>
    /// This service is to validate tenant have license on accessing date.
    ///</summary>
///<param name="tenantId">Tenant Identifier</param>
    ///<returns>True/False</returns>
publicbool ValidateTenantLicense(string tenantId)

```

Above method checks if the license start date should be greater than or equal to current date and end date should be less than or equal to current date.

```
To validate license module
///<summary>
/// This service validate the tenant have license for module.
///</summary>
///<param name="moduleNameAccessed">Module Name</param>
///<param name="tenantId">Tenant Identifier</param>
///<returns>True/False</returns>
public bool ValidateModuleLicense(string moduleNameAccessed, string tenantId)
```



For validating the usage: refer metering chapter.

Example

```
bool validPackage =
CelloSaaS.ServiceProxies.TenantManagement.TenantProxy.ValidateTenantLicense(tenantCode);
if (validPackage)
{
    //Write your logic
}
else
{
    //Show exception message to user
}
```

```
To validate package [Alternate Method]
//Get tenant license details and check valid tenant license
TenantLicense tenantLicense =
CelloSaaS.ServiceProxies.TenantManagement.LicenseProxy.GetTenantLicense(userDetails.MembershipDetails.TenantCode);
if (tenantLicense != null && !string.IsNullOrEmpty(tenantLicense.PackageId))
{
    //Write your logic
}
else
{
    //Show exception message to user
}
```

```
To check if the tenant has valid package
bool ValidateTenantLicense(string tenantId)
```

This method checks, if the license start date is greater than or equal to current date and end date is less than or equal to current date.

```
To validate modules
bool ValidateModuleLicense(string moduleName, string tenantId)
```

You need to validate the subscription before login. If valid package is true, access is allowed for user login. This is automatically taken care, if you use the default login page else the following code should be included before a user logs in.

Example

```
bool validPackage = TenantProxy.ValidateTenantLicense(tenantCode);
if (validPackage)
{
    //Write your logic
}
else
{
    //Show exception message to user
}
You can validate package another way also,
//Get tenant license details and check valid tenant license
TenantLicense tenantLicense =
LicenseProxy.GetTenantLicense(userDetails.MembershipDetails.TenantCode);
if (tenantLicense != null&& !string.IsNullOrEmpty(tenantLicense.PackageId))
{
    //Write your logic
}
else
{
    //Show exception message to user
}
```

How To Guide: [Package Management](#)

10 TENANT MANAGEMENT

Tenant Management is one of the core modules of any saas application.

Who is a tenant? If you are focused on enterprise SaaS (Business to Business applications), each of your customer would be a tenant.

Example

If you are developing a HR Payroll processing product, each company that purchases the product becomes the client AKA **tenant**. Each tenant will have Finance resource who will process payroll needs of their employees called as **Users**.

If your company use salesforce.com for sales force automation, your company is a tenant for salesforce.com.

If you are focused on B2C, each customer (you and me) will be considered as tenants.

CelloSaaS has implemented “Shared Tenancy” model, which is the best and most economical models available for managing multiple tenants. This chapter explains about the tenant management and the data model behind tenant management.

Multiple Ways to add new Tenants into the System

There are multiple ways to onboard a new tenant. They are

Using Standard User Interface to fill the customer details provision software package

One click Subscription model, where the customer will be onboarded with few information and environment will be set instantly

10.1 OnBoarding Using Admin Dashboard

This is a standard way of onboarding new tenants in CelloSaaS where the administrator of the product fill the details of the tenants and provide appropriate software package and pass on the credentials to the tenant and allow him to use the Software.

10.1.1 Add Tenant

Product admin should provide the necessary details such as Tenant, contact, address, administration, package details.

Product admin should select any one of the package for tenant, based on their need.

While provisioning Tenant, the administrator can set the number of users can be created by the tenant, If number of users is set null, then tenant can have any number of users. If the user count is set, then tenant can have specified number of users only.

Number of users should be greater than or equal to number of users for tenant.

Example: If the tenant has 10 users, then number user value will be greater than or equal to 10.

Welcome, Company Admin ! | Settings | Help | LOG OFF

CelloSaaS

[Manage Tenants](#) | [Manage Users](#) | [Manage Access](#) | [Manage Configurations](#) | [Reporting](#) | [Manage Workflow](#)

Add Tenant

Tenant Details

Tenant Code String	<input type="text"/>	Tenant Name	<input type="text"/>
Description	<input type="text"/>	Website	<input type="text"/>
URL	<input type="text"/>		

Billing Address

Address	<input type="text"/>	City	<input type="text"/>
State	<input type="text"/>	Country	<input type="select" value="Select Country"/>
Postal Code	<input type="text"/>		

Contact Details

First Name	<input type="text"/>	Last Name	<input type="text"/>
Phone	<input type="text"/>	Fax	<input type="text"/>
Email	<input type="text"/>		

Company User Details

First Name	<input type="text"/>	Last Name	<input type="text"/>
User Name	<input type="text"/>	Email	<input type="text"/>

Available Tempaltes

Available Templates	<input type="text" value="DefaultCustomTemplate"/> <input type="button" value="..."/>
---------------------	---

Package Settings

Number of Users	<input type="text"/>
Package *	<input checked="" type="radio"/> Gold <input type="radio"/> Platinum <input type="radio"/> Silver



Copyright © 2012 by techcello.com

Figure 10-1 – Add Tenant Screen

Once the tenant is created successfully, the **Tenant Admin role** will be assigned to Tenant instantly. Common roles will be copied to the newly created tenant by using Tenant Post processor method. **Theme folder is automatically created inside App_Themes folder with default style sheet.** For more information about themes refer skin configuration.

```
PostProcessProvider.Execute(tenant.TenantDetails.TenantCode)
```

10.1.2 Edit Tenant

Tenant Details

Tenant Name	audichina	Description	audichina
Website	www.audichina.com	URL	http://audichina.techcello.com
Tenant Types	TenantAdmin		

Billing Address

Address1	Address	City	
State	State	Country	UK
Postal Code	43243		

Contact Details

First Name	Admin	Last Name	audichina
Phone	234234	Fax	234234
Email	admin@audichina.com		

Package Settings

Package *	<input checked="" type="radio"/> Audi_Package <input type="radio"/> Premium Audi <input type="radio"/> Standard Audi
Number of Users	10

Cancel **Update**



Copyright © 2012 by techcello.com

Figure 10-2 – Edit Tenant Screen

Tenants Listing Page

Tenant Management

Tenant Code String	Tenant Name	Description	Website	Edit
AudiMumbaDealer	AudiMumbaDealer	AudiMumbaDealer	www.AudiMumbaDealer.com	
audiindia	Audi India	Audi India	www.audi.com	
audius	Audi US	Audi US	www.audius.com	
AudiUK	Audi UK	Audi UK	www.audiuk.com	
audiuae	Audi UAE	Audi UAE	www.audiuae.com	
audichina	audichina	audichina	www.audichina.com	

Show 10 entries Showing 1 to 6 of 6 entries

Figure 10-3 – Tenant Management Screen

10.2 One Click Application Subscription

Similarly Onboarding new customer without using the Admin User Interface can be done using the Service Methods given by celloSaaS. Developers should use these and create Tenants/Manage Tenants, all the actions possible with the User interface can be achieved using the APIs.

To Add Tenant Details

```
///<summary>
/// Insert tenant Details.
///</summary>
///<param name="tenantDetails">Tenant Details</param>
///<param name="address">Address Details</param>
///<param name="contactDetails">Contact details</param>
///<param name="tenantAdminUserdetail">User details</param>
///<param name="tenantAdminMembershipdetail">Membership details</param>
///<param name="tenantLicense">Tenant License details</param>
///<returns>Tenant Code</returns>
publicstring ProvisionTenant(TenantDetails tenantDetails, Address address, ContactDetails
contactDetails, User tenantAdminUserdetail, MembershipDetails tenantAdminMembershipdetail,
TenantLicense tenantLicense)
```

///<summary>

```
/// Insert tenant Details.
///</summary>
///<param name="tenant">Tenant Details</param>
///<param name="tenantLicense">Tenant License details</param>
///<returns>Tenant Code</returns>
publicstring ProvisionTenant(Tenant tenant, TenantLicense tenantLicense)
```

To Get tenant details

```
///<summary>
/// Get All tenant details including inactive tenants
///</summary>
///<returns>Tenant Details</returns>
publicList<Tenant> GetAllTenantInfoDetails()
///<summary>
/// Get Active Tenant Details
///</summary>
///<returns>Tenant Details</returns>
publicList<Tenant> GetTenantInfoDetails()
```

Get tenant details by tenant code

```
///<summary>
/// Get all the tenant deatils for given tenant id.
///</summary>
///<param name="tenantId">Tenant Id</param>
///<returns>Tenant Details</returns>
publicTenant GetTenantDetailsByTenantId(string tenantId)
```

To Update tenant details

```
///<summary>
/// This method is update the tenant.
///</summary>
///<param name="tenant">Tenant Details</param>
///<param name="tenantLicense">Tenant License Details</param>
///<returns>Tenant Code</returns>
publicstring UpdateTenantInfo(Tenant tenant, TenantLicense tenantLicense)
///<summary>
/// This method is update the tenant.
///</summary>
///<param name="tenantDetails">Tenant Details</param>
///<param name="address">Address Details</param>
///<param name="contactDetails">Contact Details</param>
///<param name="tenantLicense">Tenant License Details</param>
```

```
///<returns>Tenant Code</returns>
public string UpdateTenantInfo(TenantDetails tenantDetails, Address address, ContactDetails contactDetails, TenantLicense tenantLicense)
```

Example

```
//Get user details for tenant
Dictionary<string, CelloSaaS.Model.UserManagement.UserDetails> userDetails = CelloSaaS.
ServiceProxies.UserManagement.UserDetailsProxy.GetUserDetailsByTenantId(tenantId);
//Check number of users with tenant user count
if (numberOfUsers < userDetails.Count)
{
    ModelState.AddModelError("TenantStatusMessage",
    Resources.TenantResource.e_LicenseNumberOfUser);
}
```

Deactivate/Delete Tenant

```
///<summary>
/// This method is to deactivate the tenant.
///</summary>
///<param name="tenantId">Tenant Id</param>
public void DeleteTenant(string tenantId)
///<summary>
/// This method is to delete the tenant details for the input tenantid.
///</summary>
///<param name="tenantId">Tenant Id</param>
public void PermanentDeleteTenant(string tenantId)
```

10.3 Pre-Requisites for Creating Tenants

Decide where to store Application Specific Data

Create Package

List of Application Specific properties along with mandatory Information about the tenant

Provision through User Interface / Provision through Service

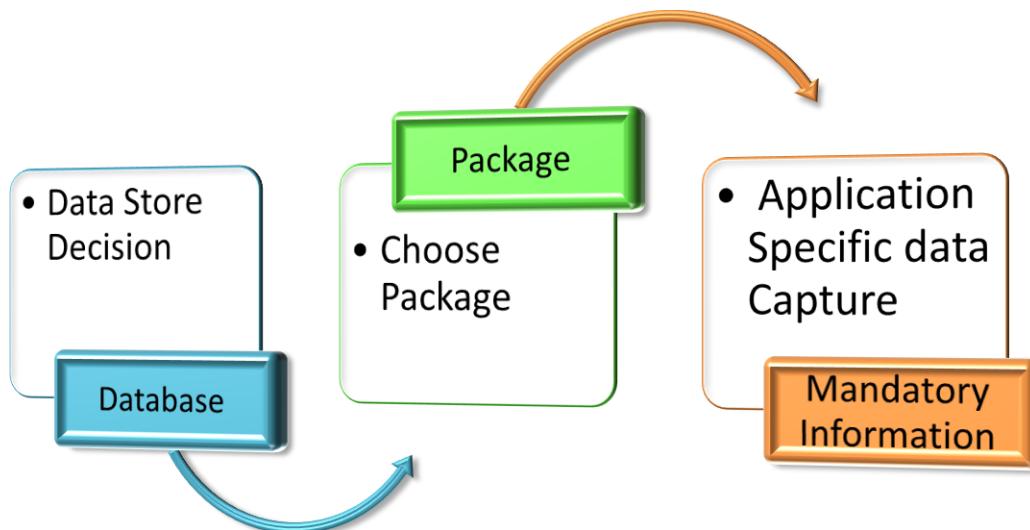


Figure 10-4 – Pre-Requisites for Creating Tenants

10.3.1 Post Process

Create Setting Templates

Roles and Privilege

Application Access via URL

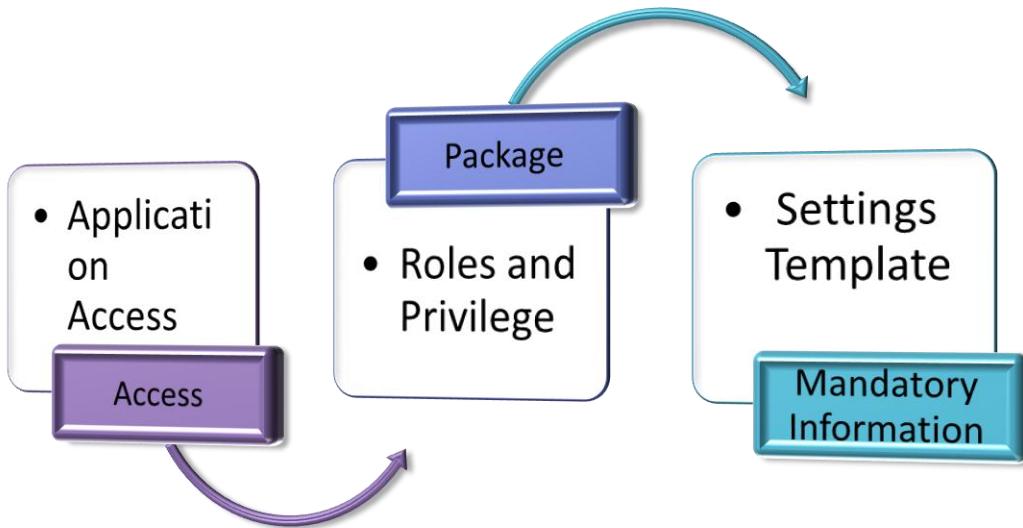


Figure 10-5 – Post Process

10.4 Tenant Hierarchy

CelloSaaS supports a hierarchical tenant structure; means a Potential Subscriber/Customer of a SaaS Product can have Sub tenants aka Child Tenants. These Sub tenants can then create Tenants as their child and so on. These Child can act as a stand alone independent Subscribers, and having rules/protocols applied by their Parent Subscribers.

The state of having customer under a customer and maintaining relationship between different entities is generally called as tenant hierarchy. In the complete hierarchical structure, the application provider is the Primary tenant of all tenants, only the application provider will be able to create the primary level tenants. These tenants can then create sub tenants under them and this will go on upto n Level and there is no restriction in the number of nodes created.

Especially in the Enterprise World of applications, the tenant hierarchy can be used very effectively without the hassle of running multiple instance of the application in the different parts of the world. The Administrator of the applications can easily spin off new nodes under primary with few clicks of

the mouse.

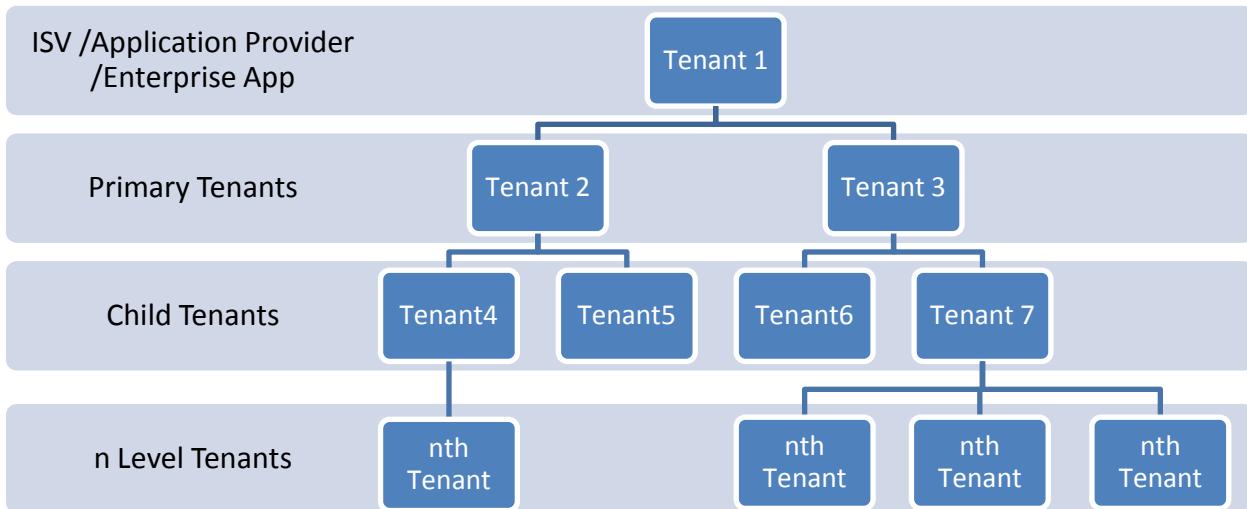


Figure 10-6 – Hierarchical Tenant System

10.5 Terminorlogy Used in Tenant Hierarchy:

Hierarch Tenant [Application Provider]

The application Provider of the SaaS product or the Administrator of the Enterprise Application becomes the Hierarchy tenant or Top most Tenant in the tree. Generally all tenants created in the application become the Children of the product owner by default.

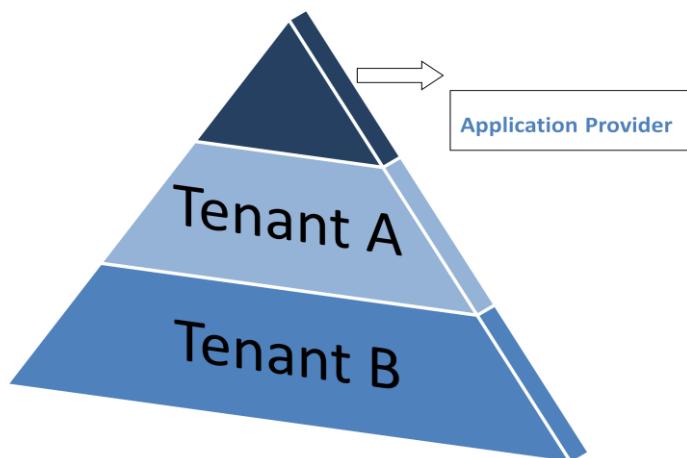


Figure 10-7 – Hierarch Tenant Flow Diagram

Primary Tenant

For the Users under Tenant1, Tenant1 is the Primary Tenant and Application provider or the Root Tenant becomes the **Super Parent**.

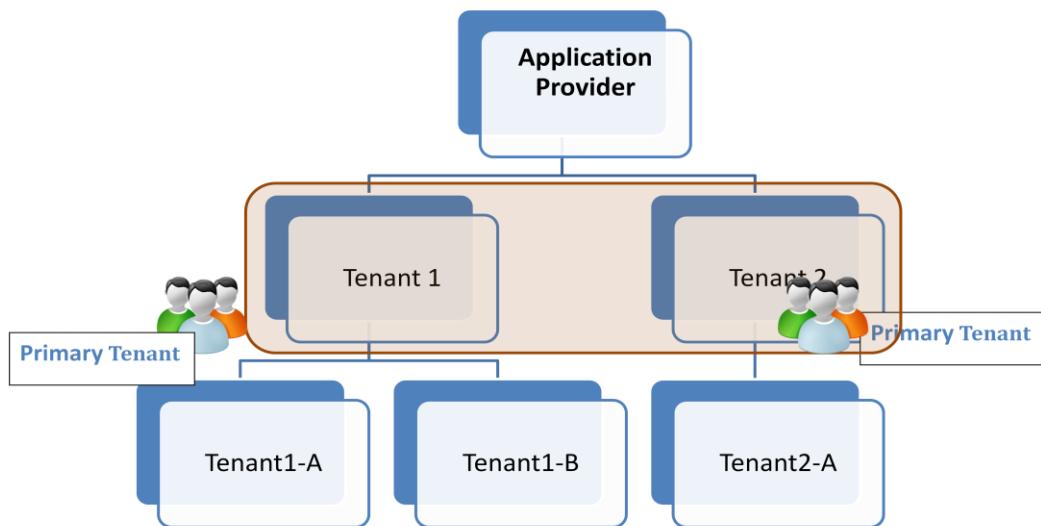


Figure 10-8 – Primary Tenant Flow Diagram

Immediate Children

The Primary tenants can create n-Number of Child Tenants; the tenants created by primary tenants are called as immediate tenants.

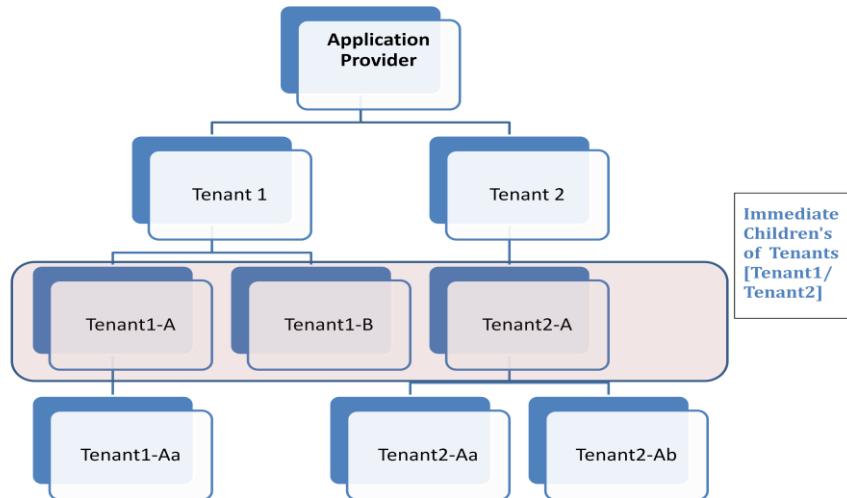


Figure 10-9 – Immediate Children Flow Diagram

All Children

Immediate Childrens and the entire children hierarchy beneath the immediate children of the tenant are totally called as **all children Tenants**.

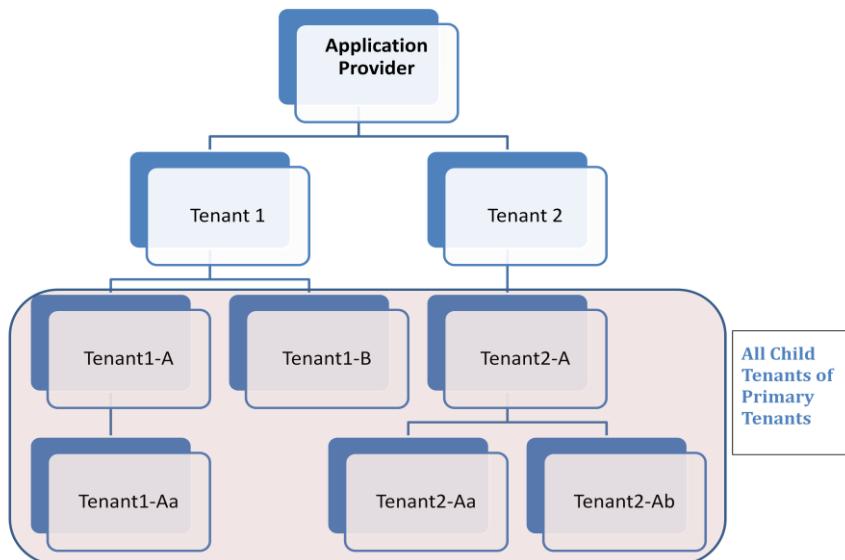


Figure 10-10 – All Children Flow Diagram

Immediate Parent

The Tenant who created the Child Tenant is called as Immediate Tenant.

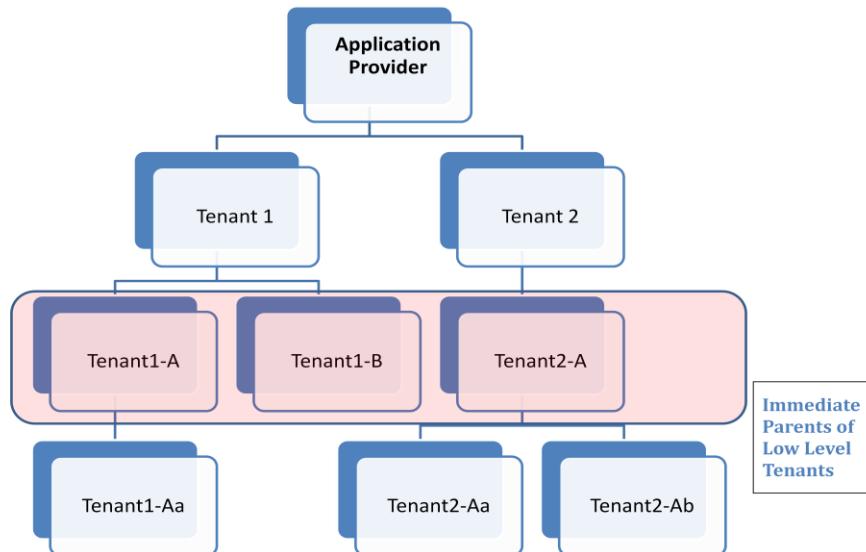


Figure 10-11 – Immediate Parent Flow Diagram

All parents

Immediate parent and the entire parent tree above the immediate parent are called as **Super Parents**

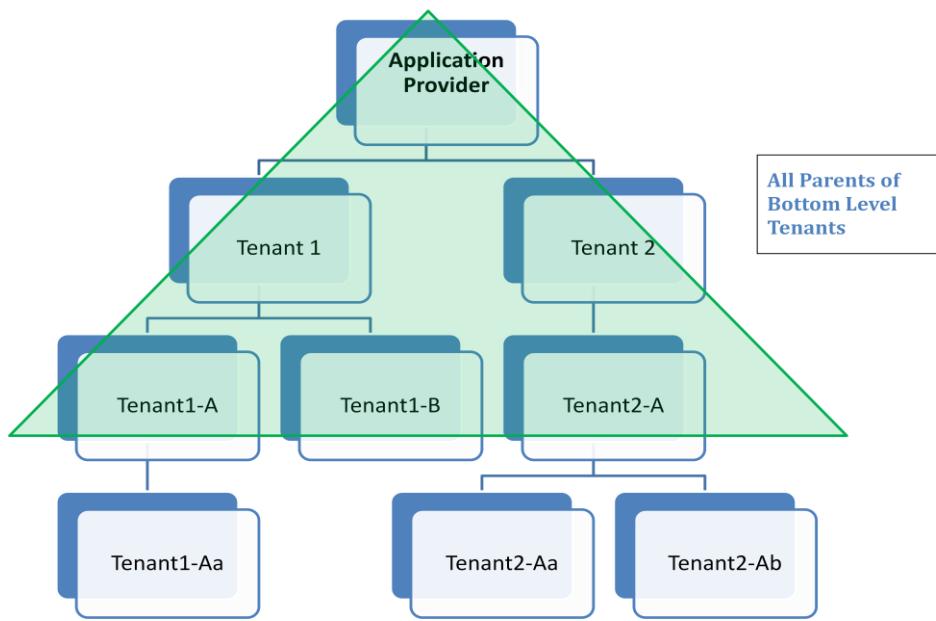


Figure 10-12 – All Parents Flow Diagram

Tenant Context

In the context of Tenant Hierarchy, a parent tenant can manage Multiple Sub tenants, similarly a **Service Administrator** can oversee or administrate a group of Tenants. When the Service admin wants to manage various actions in different Tenants, he need not logout and login as each tenant to do actions instead he can easily impersonate himself and do actions on behalf of other tenants. CelloSaaS provides impersonation capability with which the Super Admin/Service Admin can impersonate and do various activities based on the privilege and accessibility.

Primary Tenant/ Logged In Tenant

The Login screen is common for all the users of the application irrespective of their Roles/Access. It is their individual credentials and their unique company code which differentiates one user from another. When the user login, the application identifies the user against the credentials and setup a context with necessary details and this will be maintained until the user logs out.

Based on the privileges given to each user, he/she will be allowed to perform certain actions, but privileges such as Application administration, Membership management, Access Privilege will be given only to administrators.

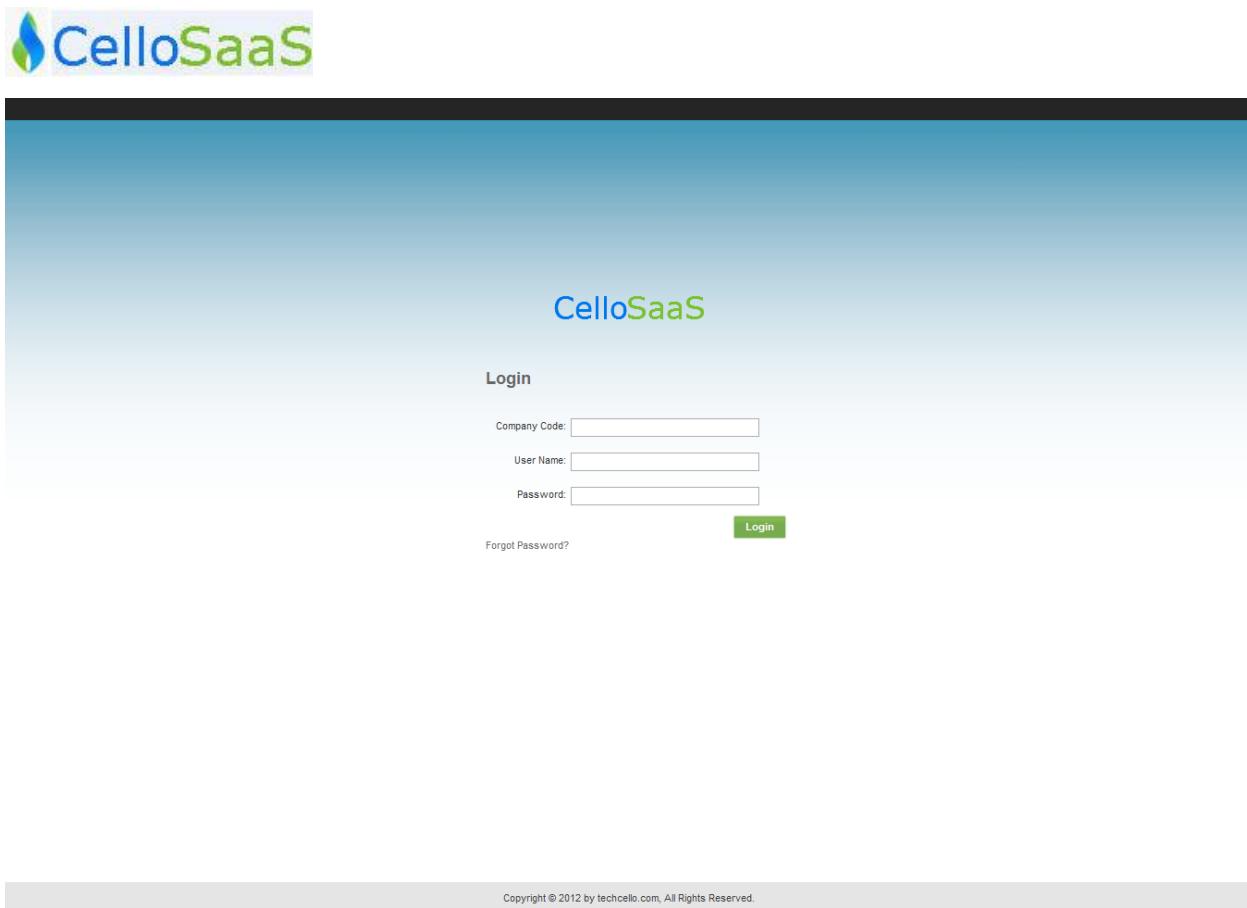


Figure 10-13 – CelloSaaS Login Screen

Session Tenant

Super Administrators/ Service Administrators are the special Roles, where in they can administrate the entire setup or manage setup of Group of tenants. When these special Administrators logs in, they gets a special Control called as **Session Switcher**, placed next to the **logout** button. This gives the list of tenants, who are under the current Administrator/Service Administrator control, he can then set the context of the session under which he would to operate operate. The tenants listed on this page are based on the user's tenant access. Any tenant for whom the user has any role gets listed there.

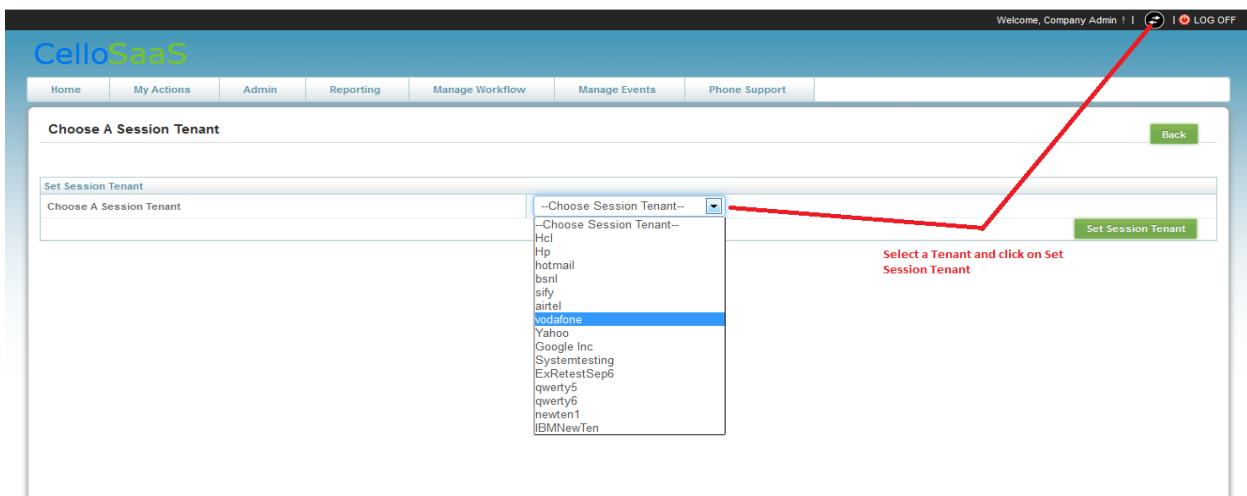


Figure 10-14 – Choose A Session Tenant Screen

Page Level Tenant Context

Welcome, Company Admin ! | LOG OFF

Tenant Management

List of approved tenants

Tenant Code String	Tenant Name	Description	Website	Tenant_ApprovalStatus	Boolean1	Date1	Float1
123	123		www.adfdfdjhjh.techcello...	Approved	False	123	Hcl
adfdfdjhjh	adfdfdjhjh		www.adfdfdjhjh.techcello...	Approved	False	323	hotmail
afdfdf	afdfdf	afdfdf	www.adfdssd.techcello.co...	Approved	False	5	Hp
airtel	airtel			Approved		432	IBMWNewTen
babylon	babylon			Approved	False	3	newtenant
BMW	bmw	bmw	www.bmw.com	Approved	True	123	abcd22
bsnl	bsnl			Approved		312	
ExRetestSep6	ExRetestSep6	ExRetestSep6	www.ExRetestSep6.techcell...	Approved	False	2323.783	0
ExRetestSep7	ExRetestSep7	ExRetestSep7	www.ExRetestSep7.techcell...	Approved	False	23232	23232
ExtendedFieldtest	ExtendedFieldtest	ExtendedFieldtest	www.ExtendedFieldtest.t...	Approved	False	23	999999 sddsds2

Show 10 entries Showing 1 to 10 of 61 entries

List of unapproved tenants

Tenant Code String	Tenant Name	Description	Website	Tenant_ApprovalStatus	Boolean1	Date1	Float1	Int1	Varchar1
ibmnew1	ibmnew1	ibmnew		WaitingForApproval	False	25	0	testibm1	
Test11	Guruu			WaitingForApproval	False	1212	0	adasasd	
TestTenant110	TestTenant110			WaitingForApproval	True	12545	2545	sdfdswf	

Show 10 entries Showing 1 to 3 of 3 entries



Copyright © 2012 by techcello.com

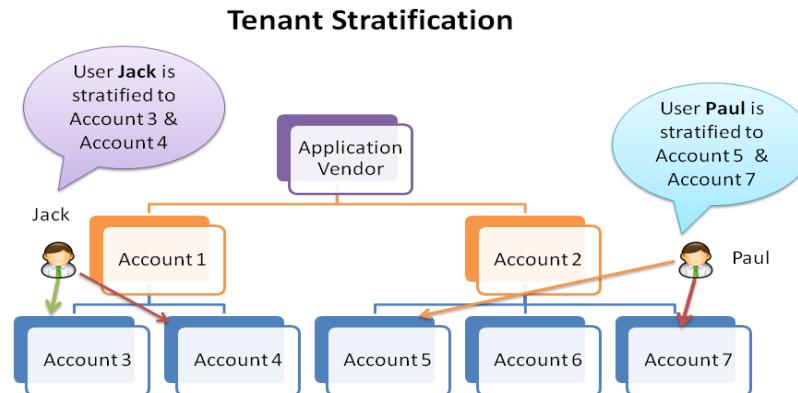
Figure 10-15 – Tenant Management Screen

In the administrative pages provided in CelloSaaS the user can operate under different tenants based on the tenant context drop down that gets listed in the admin pages. The tenants that get listed here are the tenants for whom the user has the privilege to perform that particular action.

Tenant Access

Delegated Users can access data as well as perform actions on different tenants based on the following scenarios

- Users assigned roles within the primary tenant
- User assigned a role for the child tenant. Role belongs to the primary tenant
- Users who are assigned role for the child tenants using tenant stratification. Users can perform the actions as per the privileges assigned for the roles in the parent tenant



11 METERING

Generally SaaS applications charge the customers based on the usage. Usage could be anything from a generic flat usage fee to something like no. of employees to no. of invoices processed per month. So, it is important to measure the usage per tenant.

Usage is associated with a Module. A Module can contain many Usages. The below SQL Scripts can be used to add new Usage to a Module.

CelloSaaS equips the developers to have a track on the usage. This chapter explains the mechanism implemented by CelloSaaS.

Following tables are used for keeping track of the usage.

Metering Table details:

Column Name	Data Type	Allow Nulls
Metering_ID	Varchar (50)	False
Metering_TenantID	Varchar (50)	False
Metering_UsageCode	Varchar (50)	False
Metering_UsageAmount	Decimal(18,0)	False
Metering_CreatedBy	Varchar (50)	False
Metering_CreatedOn	DateTime	False
Metering_UpdatedOn	DateTime	True
Metering_UpdatedBy	Varchar (50)	True
Metering_Status	Bit	False

Metering details are tracked in the Metering Log table.

Add connection string for metering log in web config with connection string name as 'MeteringConnectionString'.

```
<add name="MeteringConnectionString" connectionString="Data Source=192.168.2.103;Initial Catalog= Saas;User Id= Admin;Password= Admin$;" providerName = "System.Data.SqlClient"/>
```

Metering Log table structure is as follows

Column Name	Data Type	Allow Nulls
MeteringLog_ID	Varchar (50)	False
MeteringLog_UsageCode	Varchar (50)	False
MeteringLog_Amount	Decimal(18,0)	False
MeteringLog_TenantID	Varchar (50)	True

MeteringLog_LoggedBy	Varchar (50)	False
MeteringLog_LoggedDate	DateTime	False
Metering_UpdatedOn	DateTime	True
Metering_UpdatedBy	Varchar (50)	True
Metering_Status	Bit	False

Metering log details will be saved while you are saving metering details.

Metering log will be saved automatically when you are calling the increment/decrement meter usage methods.

-- Define a 'InvoiceUsage'

```
INSERT INTO [dbo].[Usage]
([Usage_Code],[Usage_ModuleCode],[Usage_UsageTypeCode],[Usage_UnitOfMeasurement]
,[Usage_Name],[Usage_CreatedBy],[Usage_CreatedOn],[Usage_Status])
VALUES
('InvoiceUsage','InvoiceModule','F27C2470-952A-45CA-808D-0DC58525B22B','1'
,'InvoiceUsage','3398F837-B988-4708-999D-D3DFE11875B3',GETDATE(),1)
GO
```

-- Add the usage to Product Tenant License Package

```
INSERT INTO [dbo].[LicensePackageUsage]
([LicensePackageUsage_UsageCode],[LicensePackageUsage_LicensePackageID]
,[LicensePackageUsage_CreatedBy],[LicensePackageUsage_CreatedOn]
,[LicensePackageUsage_Status],[LicensePackageUsage_MinCap],[LicensePackageUsage_MaxCa
p]
,[LicensePackageUsage_NumberPurchased],[LicensePackageUsage_AllowOverUsage]
,[LicensePackageUsage_IsAssignable],[LicensePackageUsage_IsPossessed])
VALUES
('InvoiceUsage','A8174FD7-808F-4CB1-A3E4-F3013684DEEE','3398F837-B988-4708-999D-
D3DFE11875B3',GETDATE(),1,0,0,1,1,1,1)
GO
```

The usages are specific to package not to a tenant. The usage Maximum Capacity can be set in the UI when creating a package (v4.2 onwards). Setting AllowOverUsage to true will not throw exception when incrementing the usage. The MinCap is not yet functionally available in the current version (v4.2).

APIs

The below API's can be used to Increment/Decrement Usage, get current usage amount and report usage logs etc.,

Namespace: CelloSaaS.ServiceContracts.LicenseManagement
Service Contract: [IMeteringService](#)

```
/// <summary>
/// This method is to increment the amount of usage for the given tenant and usage.
/// </summary>
/// <param name="tenantId">Tenant Identifier(Mandatory)</param>
/// <param name="usageName">Usage Name(Mandatory)</param>
/// <param name="amount">Usage Amount(Mandatory - Not less than or equal to Zero)</param>
```



```
/// <returns>The True/False value</returns>
/// <exception cref="ArgumentNullException">Throws ArgumentNullException if any mandatory
argument is null or empty</exception>
/// <exception cref="ArgumentException">Throws ArgumentException if any argument is producing
error</exception>
/// <exception cref="LicenseException">Throws LicenseException if any error occurs in this method
or when the usage exceeds.</exception>
bool IncrementMeterUsage(string tenantId, string usageName, double amount)
/// <summary>
/// This method is to decrement the usage amount for the given tenant and usage name.
/// </summary>
/// <param name="tenantId">Tenant Identifier(Mandatory)</param>
/// <param name="usageName">Usage Name(Mandatory)</param>
/// <param name="amount">Usage Amount(Mandatory - Not less than or equal to Zero)</param>
/// <returns>returns True/False</returns>
/// <exception cref="ArgumentNullException">Throws ArgumentNullException if any mandatory
argument is null or empty</exception>
/// <exception cref="ArgumentException">Throws ArgumentException if any argument is producing
error</exception>
/// <exception cref="LicenseException">Throws LicenseException if any error occurs in this
method</exception>
bool DecrementMeterUsage(string tenantId, string usageName, double amount)

/// <summary>
/// This method is to get the usage amount for the given tenant and usage name.
/// </summary>
/// <param name="tenantId">Tenant Identifier(Mandatory)</param>
/// <param name="usageName">Usage Name(Mandatory)</param>
/// <returns>returns UsageAmount</returns>
/// <exception cref="ArgumentNullException">Throws ArgumentNullException if any mandatory
argument is null or empty</exception>
/// <exception cref="LicenseException">Throws LicenseException if any error occurs in this
method</exception>
double GetMeterUsage(string tenantId, string usageName)

/// <summary>
/// Returns current usage amount used by the tenant.
/// </summary>
/// <param name="searchCondition">The search condition.</param>
/// <returns></returns>
public UsageSearchResult SearchUsageDetails(UsageSearchCondition searchCondition)

/// <summary>
/// Searches the usage metering log
/// </summary>
/// <param name="meteringSearchCondition">search condition</param>
/// <returns></returns>
MeteringSearchResult SearchMeteringLog(MeteringSearchCondition meteringSearchCondition)

/// <summary>
/// This method is used to reset the meter usage amount to 0.
/// </summary>
/// <param name="tenantId">Tenant Identifier</param>
/// <param name="usageCodes">Usage Codes</param>
/// <returns></returns>
bool ResetMeterUsage(string tenantId, string[] usageCodes)
```



Namespace: CelloSaaS.ServiceContracts.LicenseManagement
Service Contract: [ILicenseService](#)

```
/// <summary>
/// This method is used to validate license for usage limit of tenant.
/// </summary>
/// <param name="usageType">Usage Type(Mandatory)</param>
/// <param name="usageName">Usage Name((Mandatory)</param>
/// <param name="amount">Usage Amount(Mandatory - Not less than or equal to zero)</param>
/// <param name="tenantId">Tenant Identifier(Mandatory)</param>
/// <returns>The Usage Status</returns>
string ValidateUsageLicense(string usageType, string usageName, double amount, string tenantId)
```

Currently usageType of “BlockUsage” is only supported. Other types will be added in the next releases.

If the usage type parameter is “BlockUsage” then the above method will return status like.

WithinLimit – Usage limit is not exceeded.

OverUsageAllowed - Usage limit is exceeded but this usage has over usage limit set to true.

CrossedLimit – Usage limit is exceeded and doesn’t have over usage limit.

If method returns CrossedLimit, then you should not continue the process instead throw an exception like “Your usage limit exceeded”.

Example

```
public bool ProcessInvoice(InvoiceDetails entity)
{
    string status = LicenseProxy.ValidateUsageLicense("BlockUsage", "InvoiceUsage", 1,
UserIdentity.TenantID);

    if (!LicenseConstant.WithinLimit.Equals(status) &&
LicenseConstant.OverUsageAllowed.Equals(status))
    {
        throw new InvoiceException("Your usage limit is exceeded.");
    } else {
        // process invoice
        //
        // finally increment the usage amount
        MeteringProxy.IncrementMeterUsage(UserIdentity.TenantID, "InvoiceUsage", 1.0);
    }
}
```

Usage Report

Use the below API’s to get the current usage amount used by the tenant.

Namespace: CelloSaaS.ServiceContracts.LicenseManagement
Service Contract: [IMeteringService](#)

```
/// <summary>
/// Returns current usage amount used by the tenant.
/// </summary>
/// <param name="searchCondition">The search condition.</param>
/// <returns></returns>
UsageSearchResult SearchUsageDetails(UsageSearchCondition searchCondition)
```



Use the below API to get the list of usages made by the tenant during a billing cycle and reset the usage as necessary. Resetting the usage does not clear the previous audit log it only sets the current usage amount to 0 (zero).

```
/// <summary>
/// Searches the usage metering log
/// </summary>
/// <param name="meteringSearchCondition">search condition</param>
/// <returns></returns>
MeteringSearchResult SearchMeteringLog(MeteringSearchCondition meteringSearchCondition)
```

Simple UI are given by CelloSaaS to view the Current Usage Amount as well as the complete metering log with search functionality. These are available from Audits menu from CelloSaaS v4.2 onwards.

```
public class UsageSearchCondition
{
    /// <summary>
    /// Gets or sets the tenant id.
    /// </summary>
    /// <value>
    /// The tenant id.
    /// </value>
    public string TenantId { get; set; }

    /// <summary>
    /// Gets or sets the usage codes.
    /// </summary>
    /// <value>
    /// The usage codes.
    /// </value>
    public string[] UsageCodes { get; set; }

    /// <summary>
    /// Gets or sets the module codes.
    /// </summary>
    /// <value>
    /// The module codes.
    /// </value>
    public string[] ModuleCodes { get; set; }
}

public class MeteringSearchCondition
{
    /// <summary>
    /// Gets or sets the Tenant Identifier
    /// </summary>
    public string TenantId { get; set; }

    /// <summary>
    /// Gets or sets usag code
    /// </summary>
    public string UsageCode { get; set; }

    /// <summary>
    /// Gets or sets the identifier of user who created log
    /// </summary>
    public string LoggedBy { get; set; }
```



```
/// <summary>
/// Gets or sets the Start Date
/// </summary>
public DateTime StartDate { get; set; }

/// <summary>
/// Gets or sets the Start Date
/// </summary>
public DateTime EndDate { get; set; }

/// <summary>
/// Sort Column name
/// </summary>
public string SortString { get; set; }

/// <summary>
/// Sort Direction
/// </summary>
public string SortExpression { get; set; }

/// <summary>
/// Set Total Count of records found by the search criteria
/// </summary>
public bool SetTotalCount { get; set; }

/// <summary>
/// Page number to fetch
/// </summary>
public int PageNumber { get; set; }

/// <summary>
/// Page Size to fetch
/// </summary>
public int PageSize { get; set; }
}
```

Configuring Metering in XML

CelloSaaS offers you to configure metering feature to any of your business service through XML Configurations alone without changing the code.

Steps to configure metering for the WCF services:

For example lets take a WCF service named InvoiceService which has an operation named ProcessInvoice. We need to meter this operation by incrementing the "InvoiceUsage". Let's make changes in the configuration to achieve this.

Step 1: Open **EntityPermissions.config** file and add the below xml configuration.

```
<add name="BL">
<Entity>
<add name="YouApp.ServiceContracts.IInvoiceService">
<EntitySubElement>
<add name="ProcessInvoice" UsageCode="InvoiceUsage" UsageMode="Increment"/>
</EntitySubElement>
</add>
</Entity>
</add>
```



Step 2: Open **Web.config** file and make the below changes inside **system.serviceModel** section.

```
<extensions>
  <behaviorExtensions>
    <add name="messageInspector"
      type="CelloSaaS.Services.WCF.LoggingBehaviorExtensionElement, CelloSaaS.Services" />
    <add name="meteringInspector" type="CelloSaaS.Services.WCF.MeteringExtensionElement,
      CelloSaaS.Services" />
  </behaviorExtensions>
</extensions>

<behaviors>
  <endpointBehaviors>
    <behavior name="messageInspectorBehavior">
      <messageInspector />
      <meteringInspector/>
    </behavior>
  </endpointBehaviors>
</behaviors>
```

Add the **messageInspectorBehavior** endpoint behavior to your business WCF Service endpoints. The **meteringInspector** watches every WCF calls and whenever the configured Operation call is matched it Increments/Decrements the specified Usage.
In-Proc service calls can also be metered via configuration.

Steps to configure metering for the In-Proc service calls:

Step 1: Same as above

Step 2: Open **policyInjection.config** file and make the below changes.

```
<add name="Application Metering">
  <matchingRules>
    <add
      type="Microsoft.Practices.EnterpriseLibrary.PolicyInjection.MatchingRules.NamespaceMatchingRule,
      Microsoft.Practices.EnterpriseLibrary.PolicyInjection" name="Namespace Matching Rule">
      <matches>
        <add match=" YouApp.ServiceContracts.IInvoiceService" ignoreCase="false" />
        <add match=" YouApp.ServiceContracts.IProductService" ignoreCase="false" />
      </matches>
    </add>
  </matchingRules>
  <handlers>
    <add name="Metering Handler" type="CelloSaaS.Services.MeteringCallHandler,
      CelloSaaS.Services" />
  </handlers>
</add>
```

Add the necessary business service contracts to the above matchingRules. And create you business service instance using the below code.

```
IInvoiceService invoiceService = ServiceLocator.Resolve<IInvoiceService>();
invoiceService.ProcessInvoice(invoiceEntity);
```

Configuration based metering has its disadvantage like the usage is incremented/decrement regardless the method fails due to some reason.

12 SETTINGS

12.1 System Settings

This enables the product to manage the system level configurations in the system. Settings can be maintained based on tenant, roles and users.

Setting Meta Data is a collection of attribute details.

Setting Meta data contains Module Settings Meta Data, Attribute Set Settings Meta Data and Attribute Meta Data.

Attribute Meta Data contains attribute ID, attribute name, data field type, attribute set id, module code and feature code.

Attribute Set Settings Meta Data contains attribute set details and Attribute Meta Data.

Feature Settings Meta Data contains feature details, Attribute Set Settings Meta Data and Attribute Meta Data.

Module Settings Meta Data contains module details, Feature Settings Meta Data, Attribute Set Settings Meta Data and Attribute Meta Data.

Get settings Meta Data details by using following methods.

```
///<summary>
/// This method is used to get settings meta data details
///</summary>
///<returns>Settings meta data details</returns>
publicSettingsMetaData GetSettingsMetaData()
```

```
///<summary>
/// This method is used to get settings meta data details for tenant code
///</summary>
///<param name="tenantCode">Tenant Code</param>
///<returns>Settings meta data details</returns>
publicSettingsMetaData GetSettingsMetaData(string tenantCode)
```

12.1.1 Setting Attribute

Attribute details stored in the Setting Attribute table.

In Settings Attribute: Table ID, Name, and Data Field Type ID are mandatory fields.

Settings Attribute Table structure.

Column Name	Data Type	Allow Nulls
SettingsAttribute_ID	Varchar (50)	False
SettingsAttribute_Name	Varchar (50)	False
SettingsAttribute_DataFieldTypeId	Varchar (50)	False
SettingsAttribute_ModuleCode	Varchar (50)	True
SettingsAttribute_FeatureCode	Varchar (50)	True

SettingsAttribute_AttributeSetId	Varchar (50)	True
SettingsAttribute_CreatedOn	DateTime	False
SettingsAttribute_CreatedBy	Varchar (50)	False
SettingsAttribute_UpdatedBy	Varchar (50)	True
SettingsAttribute_UpdatedOn	DateTime	True
SettingsAttribute_Status	Bit	False

Example

SettingsAttribute_ID	SettingsAttribute_Name	SettingsAttribute_DataFieldTypeId	SettingsAttribute_ModuleCode	SettingsAttribute_FeatureCode	SettingsAttribute_AttributeSetId	SettingsAttribute_CreatedOn	SettingsAttribute_CreatedBy	SettingsAttribute_Status
Logo	Logo	0886B5C3-AC29-495B-891F-A3518080F119	NULL	NULL	NULL	2009-09-10 11:59:0 1.723	Admin	1

Module code/Feature code/Attribute set ID value can be null.

If module/feature code is null, then it is a common attribute. Otherwise attribute is specific to the module/feature.

Attributes are based on Module, Feature and Attribute Set.

Attribute Set details are stored in the Attribute Set table.

Attribute Set table structure.

Column Name	Data Type	Allow Nulls
AttributeSet_ID	Varchar (50)	False
AttributeSet_Name	Varchar (50)	False
AttributeSet_CreatedOn	DateTime	False
AttributeSet_CreatedBy	Varchar (50)	False
AttributeSet_UpdatedBy	Varchar (50)	True
AttributeSet_UpdatedOn	DateTime	True
AttributeSet_Status	Bit	False

Example

AttributeSet_ID	AttributeSet_Name	AttributeSet_CreatedBy	AttributeSet_CreatedOn	AttributeSet_Status
726BDFD4-AA40-41E5-81D6-5EB17FC3025B	Create	Admin	2009-09-10 11:59:01.723	1

You can get attribute details based on the Module code or Feature Code or Attribute Set Id.

12.1.2 Tenant Settings

Each tenant has their own settings like Logo, Theme, Master Page and etc. These settings are stored in the Tenant Settings table.

In tenant settings table Tenant ID and Attribute ID are mandatory fields.

Tenant ID and Attribute ID combination should be unique.

Tenant Settings table structure.

Column Name	Data Type	Allow Nulls
TenantSetting_TenantID	Varchar (50)	False
TenantSetting_AttributeID	Varchar (50)	False
TenantSetting_AttributeValue	Varchar (4000)	False
TenantSetting_CreatedBy	Varchar (50)	False
TenantSetting_CreatedOn	DateTime	False
TenantSetting_UpdatedBy	Varchar (50)	True
TenantSetting_UpdatedOn	DateTime	True
TenantSetting_Status	Bit	False

Example

TenantSetting_TenantID	TenantSetting_AttributeID	TenantSetting_AttributeValue	TenantSetting_CreatedBy	TenantSetting_CreatedOn	TenantSetting_Status
Company	Logo	Company_logo.jpg	Admin	2009-09-10 11:59:01.723	1

Get Tenant Settings

Get tenant settings by tenant code,

```
///<summary>
/// This method is used to get settings of a tenant
///</summary>
///<param name="tenantId">Tenant Identifier(Mandatory)</param>
///<returns>Tenant Setting details</returns>
publicTenantSetting GetTenantSettings(string tenantId)
```

Get tenant settings by tenant code and attribute set identifier:

```

///<summary>
///To get TenantSettings for a given tenantId by a specific attribute set Id.
///</summary>
///<param name="tenantId">Tenant Identifier</param>
///<param name="attributeSetId">Attribute Set Id</param>
///<returns>Tenant Settings</returns>
publicTenantSetting GetTenantSettings(string tenantId, string attributeSetId)

```

Save Tenant Settings

Add/update tenant settings attribute values using the following method.

```

///<summary>
    /// This method is used to updates the tenant settings for a tenant
///</summary>
///<param name="tenantSetting">Tenant Settings (Mandatory - Attributes , Tenant
Identifier)</param>
publicvoid UpdateTenantSettings(TenantSetting tenantSetting)

```

Delete Tenant Settings

Delete tenant setting details.

```

///<summary>
    /// This method is used to delete all tenant settings for given tenant id
///</summary>
///<param name="tenantId">Tenant Identifier(Mandatory)</param>
publicvoid PermanentDeleteTenantSettings(string tenantId)

```

12.1.3 Role Settings

Each role has its own settings. These settings are stored in the Role Settings table. In role settings: table Role ID and Attribute ID are mandatory fields. Role ID and Attribute ID combination should be unique.

Role Settings table structure.

Column Name	Data Type	Allow Nulls
RoleSetting_RoleID	Varchar (50)	False
RoleSetting_AttributeID	Varchar (50)	False
RoleSetting_AttributeValue	Varchar (4000)	False
RoleSetting_CreatedBy	Varchar (50)	False
RoleSetting_CreatedOn	DateTime	False
RoleSetting_UpdatedBy	Varchar (50)	True
RoleSetting_UpdatedOn	DateTime	True
RoleSetting_Status	Bit	False

Example

RoleSetting_RoleID	RoleSetting_AttributeID	RoleSetting_AttributeValue	RoleSetting_CreatedBy	RoleSetting_CreatedOn	RoleSetting_Status
Company\$Engineer	Theme	CelloSkin	Admin	2009-09-10 11:59:01.723	1

Get Role Settings

Get role settings by role identifier.

```
///<summary>
    /// This method is used to get role settings details for Role Id
    ///</summary>
    ///<param name="roleId">Role Identifier(Mandatory)</param>
    ///<returns>Role Setting details</returns>
publicRoleSetting GetRoleSettings(string roleId)
```

Save Role Settings

Add/Update role settings attribute values using the following method.

```
///<summary>
    /// Update role setting details
    ///</summary>
///<param name="roleSetting">Role setting details(Mandatory - Attributes, Role ID)</param>
publicvoid UpdateRoleSettings(RoleSetting roleSetting)
```

Delete Role Settings

Delete role setting details

```
///<summary>
    /// This method is used to delete all role settings for given role.
    ///</summary>
    ///<param name="roleId">Role Id(Mandatory)</param>
publicvoid PermanentDeleteRoleSettings(string roleId)
```

12.1.4 User Settings/Personalization

Each user has his/her own settings. These settings are stored in the User Settings table.

In user settings: table User ID and Attribute ID are mandatory fields.

User ID and Attribute ID combination should be unique.

User Settings table structure

Column Name	Data Type	Allow Nulls
UserSetting_UserID	Varchar (50)	False
UserSetting_AttributeID	Varchar (50)	False
UserSetting_AttributeValue	Varchar (4000)	False
UserSetting_CreatedBy	Varchar (50)	False
UserSetting_CreatedOn	DateTime	False

UserSetting_UpdatedBy	Varchar (50)	True
UserSetting_UpdatedOn	DateTime	True
UserSetting_Status	Bit	False

Example

UserSetting_UserID	UserSetting_AttributeID	UserSetting_Attribut eValue	UserSetting_CreatedBy	UserSetting_CreatedOn	UserSetting_Status
feac7ea9-c114-498c-a368-157441e67b21	Theme	CelloSkin_Red	Admin	2009-09-10 11:59:01.723	1

Get User Settings

Get user settings by user identifier

```
///<summary>
    /// To get all UserSettings for a given userId.
    ///</summary>
    ///<param name="userId">User Identifier</param>
    ///<returns>UserSettings details</returns>
publicUserSettings GetUserSettings(string userId)
```

Get user settings by user identifier and attribute set identifier

```
///<summary>
///To get UserSettings for a given userId by a specific attribute set Id.
    ///</summary>
    ///<param name="userId">User Identifier</param>
    ///<param name="attributeSetId">Attribute Set Identifier</param>
    ///<returns>User Settings</returns>
publicUserSettings GetUserSettings(string userId, string attributeSetId)
```

Save User Settings

Add/update user settings attribute values using the following method:

```
///<summary>
/// To insert/update UserSettings for a given User.
/// Check if user settings exist for a given user and attribute set id. If so update else insert.
    ///</summary>
    ///<param name="userSettings">User Setting details</param>
publicvoid UpdateUserSettings(UserSettings userSettings)
```

Delete User Settings

Delete user setting details

```
///<summary>
/// To dele all user settings for a given userId.
    ///</summary>
    ///<param name="userId">User Identifier</param>
publicvoid PermanentDeleteUserSetting(string userId)
```

12.2 User Authentication Settings

12.2.1 User Account Locking and Un-Locking

CelloSaaS supports user locking and Un-locking features out-of the box. The user will be locked in the following scenarios:

During login, if consecutive wrong password attempt made is more than the specified maximum failure attempt (tenant configurable), the user will be locked. (i.e., the user cannot log-in after this, even after specifying the correct password).

Then tenant admin or the users with required privileges can lock the users from the screen provided. The user will be locked, while attempting to reset the password in the **forgot password link** and providing wrong security answer more than the specified maximum failure attempt (tenant configurable).

When the user is locked, appropriate error message gets displayed in the log-in screen.



User lock & un-lock feature is tenant based. If the tenant setting doesn't have values, the user lock and un-lock features are ignored automatically while log-in.

The following services will provide access to these features:

```
IUserDetailsService:  
///<summary>  
/// This validates the user and update the failure attempts  
/// if the given password is wrong. If maximum allowed attempt is exceeded  
/// the account is locked for further log-in and  
/// UserLockedOutException will be thrown.  
/// UserPasswordPenultimateFailureAttempt exception will be thrown if the  
/// password attempt is final and further wrong  
/// password attempt will lock the user.  
  
///</summary>  
///<param name="userName">User Name(Mandatory)</param>  
///<param name="password">Password(Mandatory)</param>  
///<param name="tenantId">Tenant Id(Mandatory)</param>  
///<returns>True if the user is valid else False</returns>  
bool ValidateUser(string userName, string password, string tenantId);  
  
///<summary>  
/// This validates the user and update the failure attempts  
/// if the given password is wrong. If maximum allowed attempt is exceeded  
/// the account is locked for further log-in and  
/// UserLockedOutException will be thrown.  
/// UserPasswordPenultimateFailureAttempt exception will be thrown if the  
/// password attempt is final and further wrong  
/// password attempt will lock the user.  
  
///</summary>  
///<param name="userName">User Name(Mandatory)</param>  
///<param name="password">Password(Mandatory)</param>  
///<returns>True if the user is valid else False</returns>  
bool ValidateUser(string userName, string password);  
  
///<summary>  
/// This method will unlock the given array of user accounts  
/// and reset the failure attempts count  
///</summary>  
///<param name="membershipIds">array of membership ids</param>  
void UnLockUserAccounts(string[] membershipIds);
```

```

///<summary>
/// This method will lock the given array of user accounts.
///</summary>
///<param name="membershipIds">array of membership ids.</param>
void LockUserAccounts(string[] membershipIds);

/// This method checks the security answer for the given user
/// and updates the failure attempts. If exceeds the maximum
/// allowed number, then it locks the user account.
///</summary>
///<param name="tenantCodeString">tenant id of the user</param>
///<param name="userName">user name</param>
///<param name="answer">security answer</param>
///<returns>Returns True/False</returns>
bool ValidateSecurityAnswer(string tenantCodeString, string userName,
                           string answer);

```

Procedure to enable User Lock & Un-Lock feature

Add the following key to tenant setting table.

Tenant Id = tenant unique identifier

Attribute Id = "MaxPasswordFailureCount" and "MaxPasswordAnswerFailureCount"

Attribute value = positive integer (if 0 then the feature will be disabled)

You can also use the TenantSettingService to add the entry.

Sample

```

TenantSetting tenantSetting = newTenantSetting();
tenantSetting.TenantID = "B590CD25-3093-DF11-8DEB-001EC9DAB123";
tenantSetting.Settings = newSettings();
tenantSetting.Settings.Attributes = newDictionary<string, string>();
tenantSetting.Settings.Attributes.Add(AuthenticateSettingAttributeConstant.NumberO
fPasswordFailureAttempts, "3");
tenantSetting.Settings.Attributes.Add(AuthenticateSettingAttributeConstant.NumberO
fPasswordAnswerFailureAttempts, "3");
ITenantSettingsService tenantSettingsService = (ITenantSettingsService)
ServiceLocator.GetServiceImplementation(typeof(ITenantSettingsService));
tenantSettingsService.UpdateTenantSettings(tenantSetting);

```



AuthenticateSettingAttributeConstant is in CelloSaaS.Services.SettingsManagement namespace.

12.2.2 Forced Password Change

The tenant Administrator / product administrator can force the password change by selecting the users and clicking on the Force Password Expiry link in the user management view. This causes the membership entity for each user selected to be updated so that when the user logs on the system, the next time, the password will have to be changed to continue. The current and the new passwords should not be the same for both the forced password change and the expired password reset functionality and is checked against in the account controller.

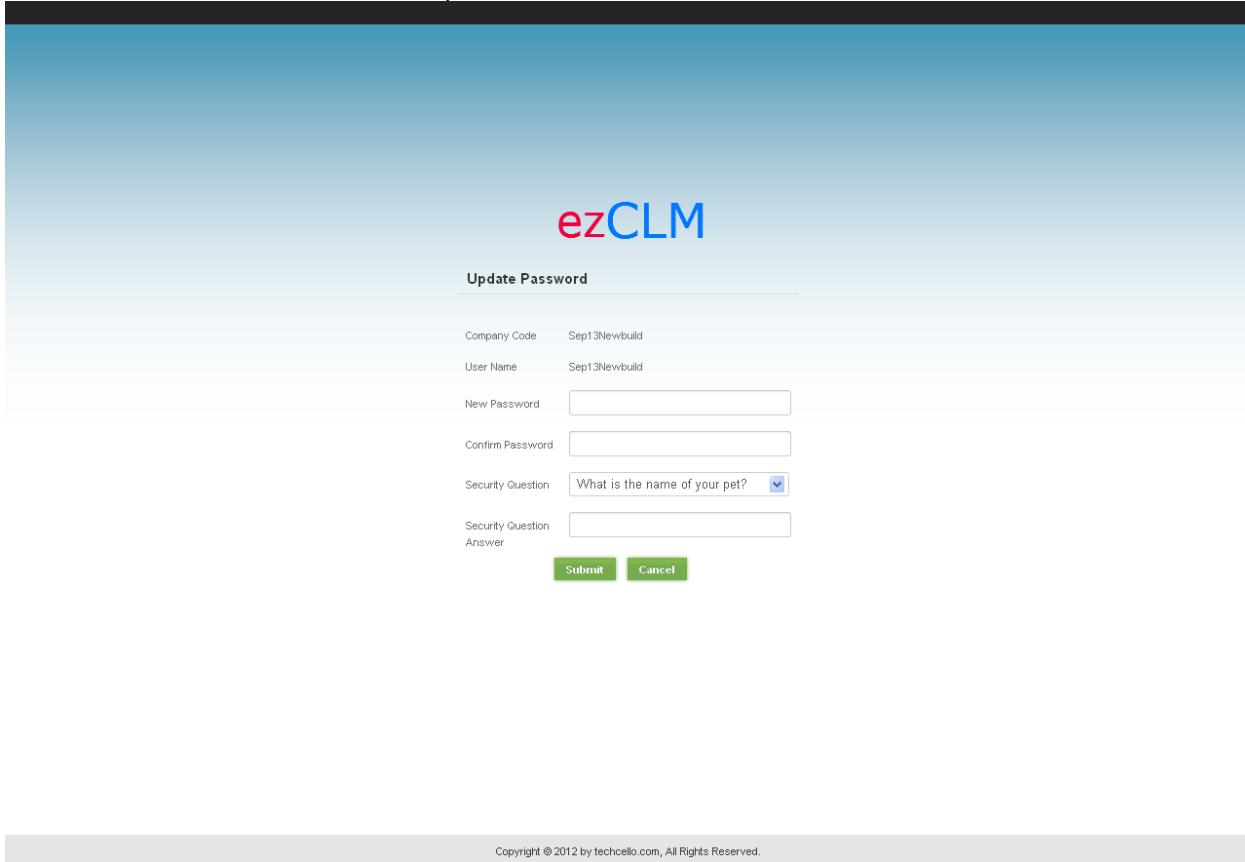
This Forced Password Change has a new privilege called as "User_ForcePasswordExpiry". This privilege is mapped to the User Entity and the ManageUser Feature of the User Entity.

12.2.3 First Time User

Whenever the new user is created, the automated PIN [company] is generated and then mailed to the user's email [based on the notification setting]. (By default password is "company"). When the user logs in to the system for the first time, the user is redirected to the view page where the user has to set the Security Question and Answer. These will be used whenever the user wants to reset the password via the "Forgot password" link at the login page.

The password entered by the user will be validated against the password validation plug-in if available, else the validation will be done as per the Regular Expression that is set in the web.config file. If there is no valid regular expression then the password validation criteria defaults to be equal to each other and of 8 characters in length.

Once the user resets the password, they will be redirected to the login page where-in they have to re-login to the system to continue further Entity Change: The Membership Details entity has another field for First time user as Membership_IsFirstTimeUser.



The screenshot shows the 'Update Password' form from the ezCLM application. The form includes fields for Company Code (Sep13Newbuild), User Name (Sep13Newbuild), New Password, Confirm Password, Security Question (set to 'What is the name of your pet?'), and Security Question Answer. There are 'Submit' and 'Cancel' buttons at the bottom. The background features a blue gradient header with the 'ezCLM' logo.

Figure 12-1 – Tenant Management Screen

12.2.4 UserPasswordExpiration [Optional]

This feature is enabled on a per-tenant basis and the number of days the passwords to be expired will be set from the tenant settings view page by the corresponding tenant. When a tenant updates / sets the password expiration days

The existing user's password expiration date [calculated based on the last logon date from membership entity] will be updated.

The status of the expiry can be obtained via the Expiration_Status field of the UserPasswordExpiration Entity.

The UserPasswordExpiration Entity also tracks the history of the password changes / resets that are being made by the user. Every time a password reset is made, the password expiry setting is maintained as history by setting the Expiration_Status to 0. Then a new record is created with the Expiration_Status to 1 for the user.



This feature can be localized tenant wise by maintaining the complete user details and managing the data via the User Connection String Setting.

The User Password Expiry will be updated every time the user resets or changes the password.

Example via the Forgot Password, Reset Password, forced password change

During the login, there is a check on whether the user password was expired or not. If so the user will be redirected to a Reset Expired Password Page.

This expiration process is scheduled via the Quartz scheduler service (Attached in the release)

package) so that the password expiration dates will be updated as per the scheduled process.

Whenever a tenant sets the password expiration days to be 0, the password expiration feature will be disabled for that tenant.

Figure 12-2 – User Management Screen

12.2.5 PasswordValidationPlugin

Settings in the Web.Config File

The following settings are to be added in the App setting sections:

ValidationExpression: A regular expression that can be used to validate the strength of the input password. Example : "(?!^[\d]{8,10})^(?!.*\s)(?!.*[\W])^([a-zA-Z\d]{8,10})\$" is the default regular expression employed by cellosaas.

ValidationFailureMessage: Here we have to set the message to be displayed to the user when the password validation fails. This message should reflect the criteria on which the password validation failed, i.e. which criteria for the password validation is failed.

ValidationSuccessMessage: Here we have to set the message to be displayed to the user when the password validation succeeds.

If you would like to implement a custom password validation service, create a new service that inherits from the IPasswordValidationService.

The following methods have to be implemented in the custom password validation service, PasswordValidationStatus ValidateUserPassword(string newPassword, string confirmPassword);

PasswordValidationStatus ValidatePassword(string password);

The Class PasswordValidationStatus has the following members:

bool IsPasswordValid

string Message

12.2.6 Hash/Encrypt mechanism for password

The option of choosing the encryption mode for the passwords [including the WCF Shared Key] has to be mentioned in the web.config file.

The Web.config file should have the following appSettings key-value pair.

```
<addkey="PasswordFormat" value="Encrypted"/>
```

The possible values are: Hashed (OR) Encrypted. The Hashing Service should be mapped to the password encryption provider in the unity.config file.

```
<typeAliasalias="PasswordHashingService" type="CelloSaaS.HashProvider.PasswordHashingService, CelloSaaS.HashProvider"
```

and

```
<typetype="IPasswordEncryptionService" mapTo="PasswordHashingService" name="IPasswordEncryptionService"></type>
```

Whenever the password mode is not set in the web.config file, the mode defaults to Encryption.



The password mode should not be changed from hashed to encrypted as the hashed passwords cannot be recovered.

The MembershipDetails will contain the hashed password as such and any attempt to Decrypt the Hashed password will throw the NotSupportedException. Also the email based notifications will not have the password in the content when the encryption mode is set to hashed as the hashed password is not decryptable.

12.2.7 Multiple User and Authentication table

If a tenant wants to have User and membership details in a separate database, this feature allows to set tenant based user connection string.

To enable this feature, we have to change the DALConnectionString Provider in web.config file.

```
<dalConnectionStringManagerdefaultProvider="Default">
<providers>
<addname="Default"assembly="CelloSaaS.Services" type="CelloSaaS.Services.Providers.UserConnectionStringProvider"/>
</providers>
</dalConnectionStringManager>
```

In case a separate database to be maintained for the User Details, then the appSettings to be set as

```
<addkey="DalConnectionStringManager" value="UserConnectionStringProvider"/>
```



Whenever a tenant is created by the product admin, there is an option for the tenant to be created with a user connection string. This connection string should contain the following table (Scripts are attached in the release package)

Membership

MembershipHistory

Address

AddressExtn

AddressExtnValues

UserDetails

UserDetailExtn

UserDetailExtnValues

UserRoles

UserSetting

UserPasswordExpiration

At the time of the tenant creation, tenant's user details are stored in the database pointed by the user connection string setting. This setting is not changeable or editable after the tenant is created. The data that is being stored in the database tables in the user connection string should adhere to the same schema as that of the CelloSaaS database.

Cross tenant queries will be not supported for the tenant with the user connection string setting. The following methods in the user details service will not be supported due to this user connection string setting.

GetUserCountByTenantIds

SearchAllUserDetailsInOtherTenants

SearchUserDetailsInOtherTenants

GetUsersInRole

GetUsersInRoles

GetUserCountByTenantIds

Further, in the role service, the following methods will not be supported

GetUsersInRole

GetUsersInRole



Please note that the Database Provider for both the CelloSaaS connection string and the user connection string should be the same.

If this feature is not needed, then you have to add the following connection string with CelloSaaS connectionstring details.

```
<addname="UserConnectionString"connectionString="DataSource=.\SQLEXPRESS;Initial Catalog=CelloSaaS2.23RC;User Id=sa;Password=cello@123;"providerName = "System.Data.SqlClient"/>
```

Welcome, Admin Audi | Settings | Help | LOG OFF

CelloSaaS

Manage Tenants | Manage Users | Manage Access | Manage Configurations | Reporting | Workflow | Phone Support

Edit Tenant

Tenant Details

Tenant Name	<input type="text" value="audichina"/>	Description	<input type="text" value="audichina"/>
Website	<input type="text" value="www.audichina.com"/>	URL	<input type="text" value="http://audichina.techcello.com"/>
Tenant Types	<input type="text" value="TenantAdmin"/>		

Billing Address

Address1	<input type="text" value="Address"/>	City	<input type="text" value="City"/>
State	<input type="text" value="State"/>	Country	<input type="text" value="UK"/>
Postal Code	<input type="text" value="43243"/>		

Contact Details

First Name	<input type="text" value="Admin"/>	Last Name	<input type="text" value="audichina"/>
Phone	<input type="text" value="234234"/>	Fax	<input type="text" value="234234"/>
Email	<input type="text" value="admin@audichina.com"/>		

Package Settings

Package *	<input checked="" type="radio"/> Audi_Package	<input type="radio"/> Premium Audi	<input type="radio"/> Standard Audi
Number of Users	<input type="text" value=""/>		

Cancel **Update**

Figure 12-3 – Edit Tenant Screen

12.2.8 Membership History

The MembershipHistory entity has been created to keep track of the past membership details. This is a replica of the membership entity. This entity will be updated for each change in the membership entity. Only the current data in the membership will be available in the membership entity and all of the previous states will be tracked in the membership history entity alone.

12.2.9 User Description

The new field is added to the User Entity called as User_Description. This field is a non-mandatory field.

12.2.10 Instrument User Login and User Logout

The User Login and Logout actions can be logged with the help of the Event activity logging feature. These events are to be called from the Account controller's login and logoff methods to enable the tracking of the user login and logout activities. The logout will be tracked only if the user logs off the application via the logout link in the application. If the browser is closed, the logout activity will not be stored for the logged in user.

The logs are maintained in the ActivityLogs table in the CelloSaaS Database.

12.2.11 Lock, Unlock Users

The tenant Administrator / product administrator can select all the users and lock them or unlock them on a single user or as a collection of users from the Manage User Page.

These two features are mapped to the User Entity and the Manage User Feature and their corresponding privileges are User_Lock and User_UnLock.

12.2.12 Quartz Scheduler

The quartz scheduler is enabled to process the userpasswordexpirystatus api of the UserDetailsService so that the Userpasswordexpiration is forced for all the users across all the tenants. In case of any exception in accessing the database of the userconnectionstring set by a particular tenant, the exception details are logged and can be consulted for suitable actions. The job will run everyday and updates the expiry status for all tenant users. It is configured as Quartz Job. The installer package provided will install the Service under "Program Files/TechCello/CelloSaaS Quartz Server".

Change the connection string in the config file: (Mandatory)
Quartz.Service.exe.config:

```
<add name="ApplicationConnectionString" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=CelloSaaSDb;User Id=user;Password=password;" providerName="System.Data.SqlClient"/>
<add name="CelloSaaSConnectionString" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=CelloSaaSDb;User Id=user;Password=password;" providerName="System.Data.SqlClient"/>
```

To change the job runtime interval, edit the quartz_jobs.xml file:

<repeat-interval>86400000</repeat-interval> --> value in seconds or you can add the corn like entry supported by quartz .net.

For more information on setting up job triggers refer <http://quartznet.sourceforge.net/>

The Migration Script has both the Schema and the data level changes to the CelloSaaS Database and the creation of the UserConnectionString Database.

12.2.13 User Activity

Allowing only single active session per user

Security of enterprise applications largely depends on the architecture framework. CelloSaaS framework plays a very vital role in securing enterprise applications authentication and authorization layer.

Often users may try to login with their credentials from different computer, or multiple users with the same credentials accessing the application, since the distributed application can be accessed from anywhere and everywhere. It is important to control the authentication system of the application to allow only valid users as well as users who are not logged in already from a system, because at any point there cannot be two active session for a single user.

CelloSaaS framework offers customizability

To change the authentication pattern as required.

To configure single session per user or multiple session per user.

Benefits

- Keeps track of the user
- Controls the access and usage of the application
- Analyzes the actual usage of the application
- Allows limited users to access the application

Configuration Settings

This feature can be turned **ON** [true] and **OFF** [false] via the Application Settings in Web.config like the one given below,

```
<appSettings>
  <!-- Enable UserActivity Tracing. Possible Values are [true / false]-->
  <add key="EnableUserActivityTrace" value="true"/>
</appSettings>
```

Methods	Description
DateTime AddUserActivity(string userId)	Add logged in user into the persistent layer for reference
int UpdateUserActivity(string userId)	Updates the user activity along with timestamp and other details
int DeleteUserActivity(string userId)	Deletes the user details from the persistent layer when they logout or when the actual session time out expires
UserActivity GetUserActivity(string userId)	Get the current user activity from the persistent layer
long GetActivitySpan(string userId)	Get the current user activity time span from the persistent layer
bool IsSessionActive(string userId)	Check whether the user's session active before authorizing the user into the system

Scope for Extensions

Sealed functionality

Limitations

In case of any accidental closure of the browser, the user has to wait till the set timeout to continue logging in and using the application.

```
if (ConfigHelper.CanTraceUserActivity())
{
    if (UserActivityProxy.IsSessionActive(userDetails.User.UserId))
    {
        this.ModelState.AddModelError("login",
            Resources.AccountResource.e_UserActivity_SessionActive);
        return this.View();
    }
    UserActivityProxy.AddUserActivity(userDetails.User.UserId);
}
```

12.2.14 Account Linking and Sharing

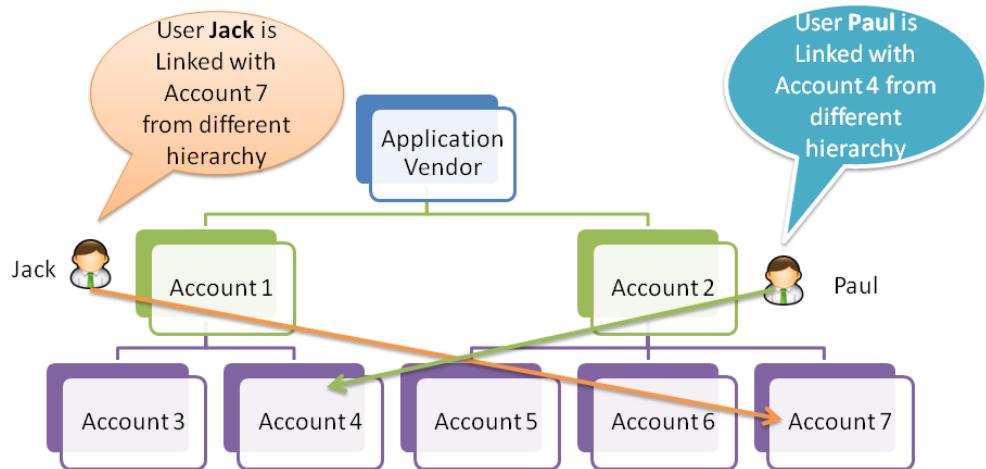
Sharing users between multiple tenants [Being Users of Multiple Tenants]

User-Sharing

A user will be created under a tenant and he/she will be an integral part of the particular tenant alone. But sometimes there are business requirements which force a single user to be shared among multiple Tenants. For example, if you are building a loyalty management system and you may want to have two of your grocery stores share the same customer. This is optional and there is no enforcement that all the tenants and users have to share the users across.

To link a user from a tenant across hierarchy as shown in the image, use link user facility to link users across hierarchy. Once linked the respective tenants can assign different class/role, based on which user will be given with the features or functionalities.

User Linking



CelloSaaS User sharing across Multiple Tenants

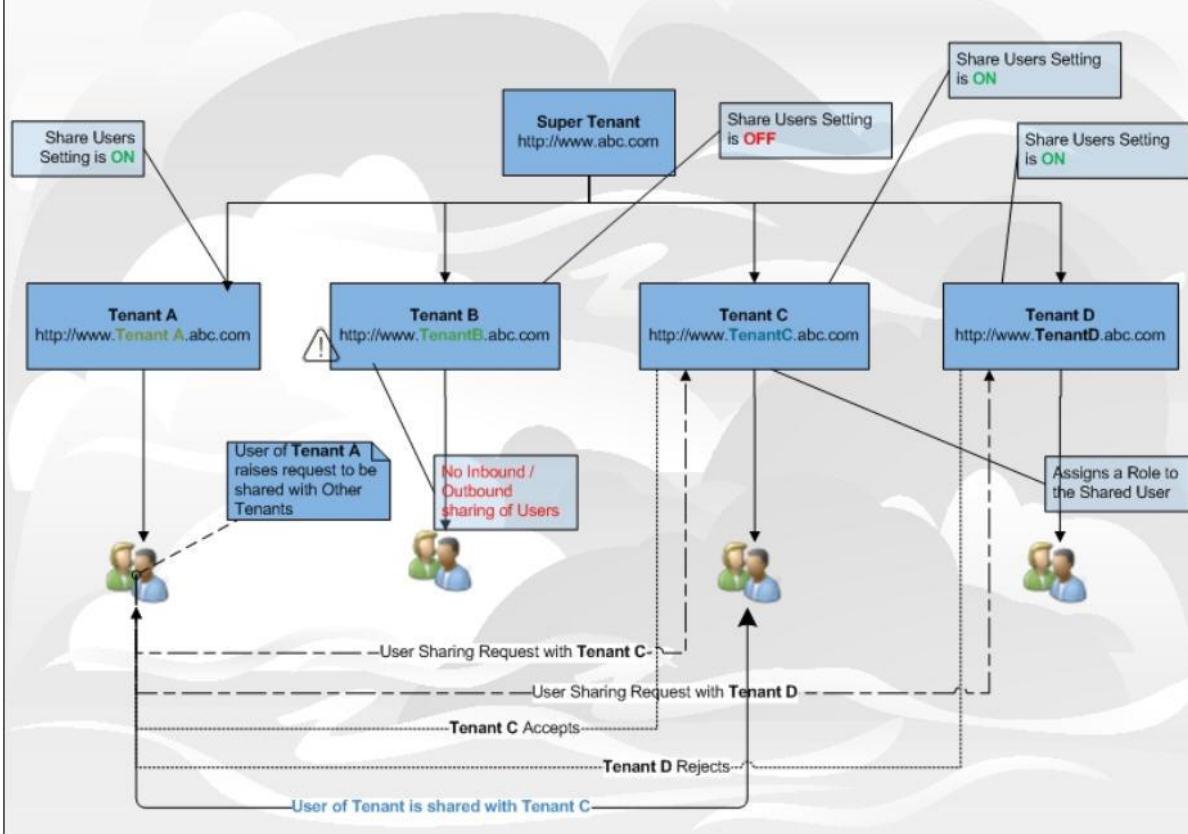


Figure 12-4 – User Sharing Screen

12.2.15 User Requests the Tenant

User1 of Tenant1 will set its attribute to be Open/Sharable [LinkByOtherTenantUsers]. These kinds of Open/Sharable users from various tenants will be displayed to other tenants. The Interested Tenants can merge these users to their Administration by raising a request. Once they apply for merge, an approval request will be sent to the appropriate Tenant Admin. The Particular Tenant Admin acknowledges the request, the response will be sent back to the Requested Tenant, and then the user will be shared to the Tenant.

12.2.16 Tenant Request the User

If the user has any unique identifier such as email/ SSN, the details of the user are entered. If there is already a match in the system for that user identifier, system can prompt whether they can add that user under them. If the tenant chooses to add the same user, it is sent as an approval which is then approved by the user.

Methods

```
Dictionary<string, LinkedTenantUsers> GetAllLinkedTenantUsers(string tenantId)
Dictionary<string, LinkedTenantUsers> GetAllLinkedTenantDetails(string userId)
Dictionary<string, LinkedTenantUsers> GetLinkedTenantUsers(string tenantId, RequestStatus
requestStatus)
Dictionary<string, LinkedTenantUsers> GetLinkedTenantsDetails(string userId, RequestStatus
requestStatus)
string LinkOtherTenant(LinkedTenantUsers linkedTenantUsers)
void UpdateLinkedTenantsUser(LinkedTenantUsers linkedTenantUsers)
```

LinkedTenantUsersModel

```
public class LinkedTenantUsers : BaseEntity
{
    ///<summary>
    /// Gets or sets the id.
    ///</summary>
    ///<value>The id.</value>
    public string Id { get; set; }
    ///<summary>
    /// Gets or sets the tenant id.
    ///</summary>
    ///<value>The tenant id.</value>
    public string TenantId { get; set; }
    ///<summary>
    /// Gets or sets the user id.
    ///</summary>
    ///<value>The user id.</value>
    public string UserId { get; set; }
    ///<summary>
    /// Gets or sets the tenant details.
    ///</summary>
    ///<value>The tenant details.</value>
    public TenantDetails TenantDetails { get; set; }
    ///<summary>
    /// Gets or sets the user details.
    ///</summary>
    ///<value>The user details.</value>
    public UserDetails UserDetails { get; set; }
    ///<summary>
    /// Gets or sets the request status.
    ///</summary>
```

```

///<value>The request status.</value>
publicRequestStatus RequestStatus { get; set; }
///<summary>
/// Gets or sets the comments.
///</summary>
///<value>The comments.</value>
publicstring Comments { get; set; }
///<summary>
/// Gets or sets the request by.
///</summary>
///<value>The request by.</value>
publicstring RequestedBy { get; set; }
///<summary>
/// Gets or sets the created by.
///</summary>
///<value>The created by.</value>
publicstring CreatedBy { get; set; }
///<summary>
/// Gets or sets the created on.
///</summary>
///<value>The created on.</value>
publicDateTime CreatedOn { get; set; }
///<summary>
/// Gets or sets the updated by.
///</summary>
///<value>The updated by.</value>
publicstring UpdatedBy { get; set; }
///<summary>
/// Gets or sets the updated on.
///</summary>
///<value>The updated on.</value>
publicDateTime UpdatedOn { get; set; }
///<summary>
/// Gets or sets a value indicating whether this <see cref="LinkedTenantUsers"/> is status.
///</summary>
///<value><c>true</c> if status; otherwise, <c>false</c>.</value>
publicbool Status { get; set; }
}

```

The Request Status has Five status as follows

```

publicenumRequestStatus
{
    Open,
    WaitingForTenantApproval,
    WaitingForUserApproval,
    Approved,
    Rejected
}

```

Open	Open for send Request
Waiting for Tenant Approval	User sent the request to tenant and waiting for tenant admin approval
Waiting for User Approval	Tenant Admin sent the request to user and waiting for user's approval
Approved	Request was approved

Rejected

Request was rejected

GetAllLinkedTenantUsers()

It takes *tenantId* as parameter and lists all the linked users details based on the tenantId with all status [waiting for Approval, Approved, and Rejected].

GetAllLinkedTenantDetails()

It takes *UserId* as parameter and lists all the tenant details which has been enabled the settings called *IsLinkedByOtherTenantUsers* with this users from all different status above.

GetLinkedTenantUsers()

It takes *tenantId* and the Request Status as parameters and lists the *LinkedUsers* details with the given status for this *tenantId*.

E.g.: List out only the approved users details

GetLinkedTenantsDetails()

It takes *UserId* and the Request Status as parameters. It gets the *LinkedTenant* details with the given status for this *userId*.

E.g.: List out only the approved tenant complete details

LinkOtherTenant()

Pass *LinkedTenantUsers* as a parameter. It is used to user request to the tenant or tenant request to the user.

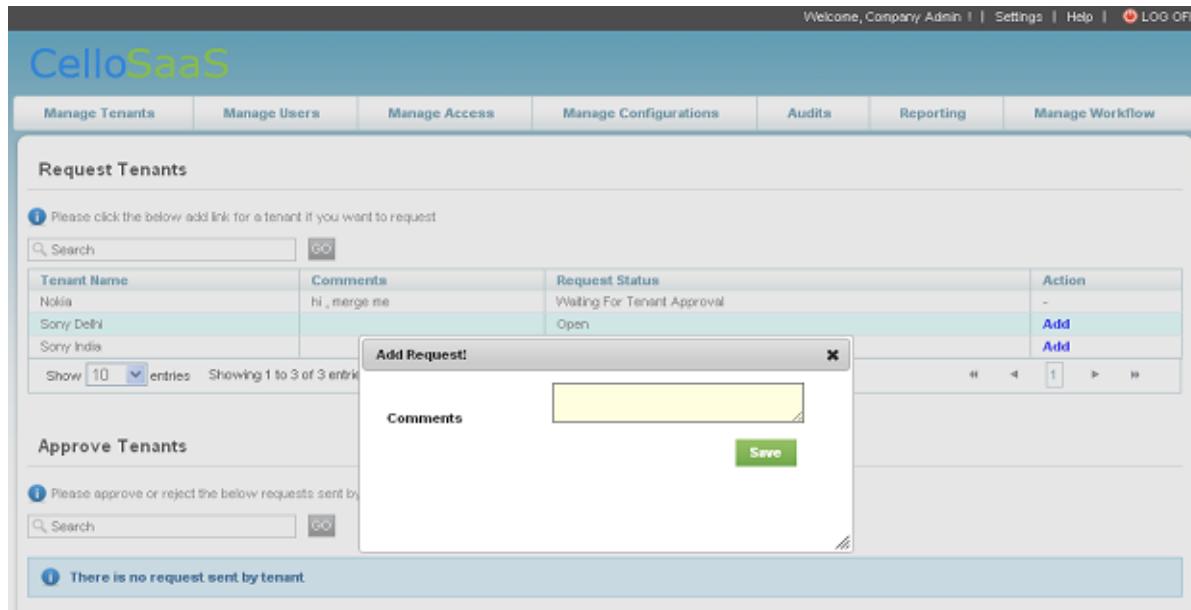
E.g : User Send the request to the tenant.

UpdateLinkedByOtherTenantsUser()

Pass *LinkedTenantUsers* as a parameter to approve or reject the tenant or user request.

E.g: Tenant approve or reject the user request

Request Tenants



The screenshot shows the 'Request Tenants' section of the CelloSaaS application. At the top, there is a navigation bar with links for Manage Tenants, Manage Users, Manage Access, Manage Configurations, Audits, Reporting, and Manage Workflow. The 'Manage Tenants' link is highlighted.

The main area is titled 'Request Tenants'. It contains a table with columns: Tenant Name, Comments, Request Status, and Action. There are three entries in the table:

Tenant Name	Comments	Request Status	Action
Nokia	hi , merge me	Waiting For Tenant Approval	-
Sony Delhi		Open	Add
Sony India			Add

Below the table, there is a search bar and a message: 'Please click the below add link for a tenant if you want to request'. A 'Show 10 entries' dropdown is also present.

On the right side, there is a modal window titled 'Add Request' with a 'Comments' input field and a 'Save' button. Below the table, there is another section titled 'Approve Tenants' with a search bar and a message: 'Please approve or reject the below requests sent by'. A message at the bottom states: 'There is no request sent by tenant.'

Figure 12-5 – Requests Tenant Screen

Approve User's Requests / Request User

Welcome, Company Admin | Settings | Help |  LOG OFF

CelloSaaS

Manage Tenants Manage Users Manage Access Manage Configurations Audits Reporting Manage Workflow

Approve User

1 Please approve or reject the below user's requests

User Name	First Name	Comments	Request Status	Action
admin	admin	resubmitted	Approved	Reject

Show 10 entries Showing 1 to 1 of 1 entries

Requested users

1 Following users are requested by this tenant

User Name	First Name	Comments	Request Status
admin	admin	linked with company	Waiting For User Approval

Show 10 entries Showing 1 to 1 of 1 entries

Figure 12-6 – Approve User Screen

13 DEVELOPER PRODUCTIVITY

The key to Techcello development productivity is that while much of the code (at least 50% or more) is automatically generated based on the metadata entered by the developer, the developer is still able to customize any part of the code that is generated. This customization is possible without having to worry about losing a customization to any tier or layer -- including those to stored procedures.

13.1 Code Generator

Code Generator is a template-based effective feature which reduces the developer's time and effort in writing the mundane code for Entities, Model, Data Access Layer, Services, DAL, Controllers, designing Views, forms and so on. With Code Generator, developers will not only benefit from the increased productivity, they will actually concentrate more on the core business domain knowledge, data modelling, user experience, business logic, and workflow of the application.

13.1.1 Features

- Generates C# Code as per Cello Framework
- Generates Code for all the layers of the application,, i.e. Model,DAL,Services, Web etc
- Generates classes as per the table structure
- Injects Security validations at the Data access Layer
- Injects Data scope policies at the Data access Layer
- Creates necessary Configuration code for registering the list fields in Cello grid & form
- Emits well commented code and follows the best practices
- Creates necessary scripts to register the module/feature in the Cello Meta database
- Creates Controller class and aspx page
- Developer gets the complete control over the emitted code, they can go ahead and write append/modify the code.

13.1.2 Prerequisites

The ASP.NET MVC 3 run-time components require the following software:

.NET Framework version 4.

The ASP.NET Web Pages run-time feature (AspNetWebPages.msi). The ASP.NET MVC 3 installer now depends on the installation of this feature, which contains the assembly that implements the Razor syntax.

ASP.NET MVC 3 Visual Studio 2010 tools require the following software:

Visual Studio 2010 or Visual Web Developer 2010 Express.

ASP.NET MVC 3.0 & Tools Update

Database and IIS

SQL Server 2005 or above

IIS 6.0 or above

CelloSaaS Specific

Enterprise Library for .NET Framework 4.

Visual Studio 2010 Software Development Kit (SDK).

Guidance Automation Extension (GAX) 2010.

Guidance Automation Toolkit (GAT) 2010.

Installation Notes for MVC 3.0

The ASP.NET MVC 3 RC for Visual Studio 2010 can be downloaded from the following page:

<http://go.microsoft.com/fwlink/?LinkId=191797>

ASP.NET MVC 3 can be installed and can run side-by-side with ASP.NET MVC 2. However, you must uninstall ASP.NET MVC 3 Preview 1 or ASP.NET MVC 3 Beta before installing ASP.NET MVC 3 RC.

More information about MVC refer the link below

<http://www.asp.net/mvc/mvc3>

Sample Web applications to quickly get started with Asp.Net MVC

<http://www.asp.net/tutorials/older-versions/getting-started-with-mvc/getting-started-with-mvc-part1>

Documentation

Documentation for ASP.NET MVC is available on the MSDN Web site at the following URL:

<http://go.microsoft.com/fwlink/?LinkId=205717>

13.1.3 Components of code generator

The below two files will be shipped along with the CelloSaaS Framework. These files contains the necessary files to install and run the code generator

CelloSaaS.ProductivityTool_CelloSaaS.DslPackage.vsix
CelloSaaSCCodeGeneration.zip

Getting Started

To create a new Solution, follow the process given below. This will create an empty solution with the necessary layered architecture, folder structure, Web Project, Code Generator etc

- ☞ Open Visual Studio
- ☞ Click on New project → Select Visual C# → Select CelloSaaSCCodeGeneration Template
- ☞ Type in the Name of the Project
- ☞ Select the Location to save
- ☞ Type in the Solution Name
- ☞ Click on OK

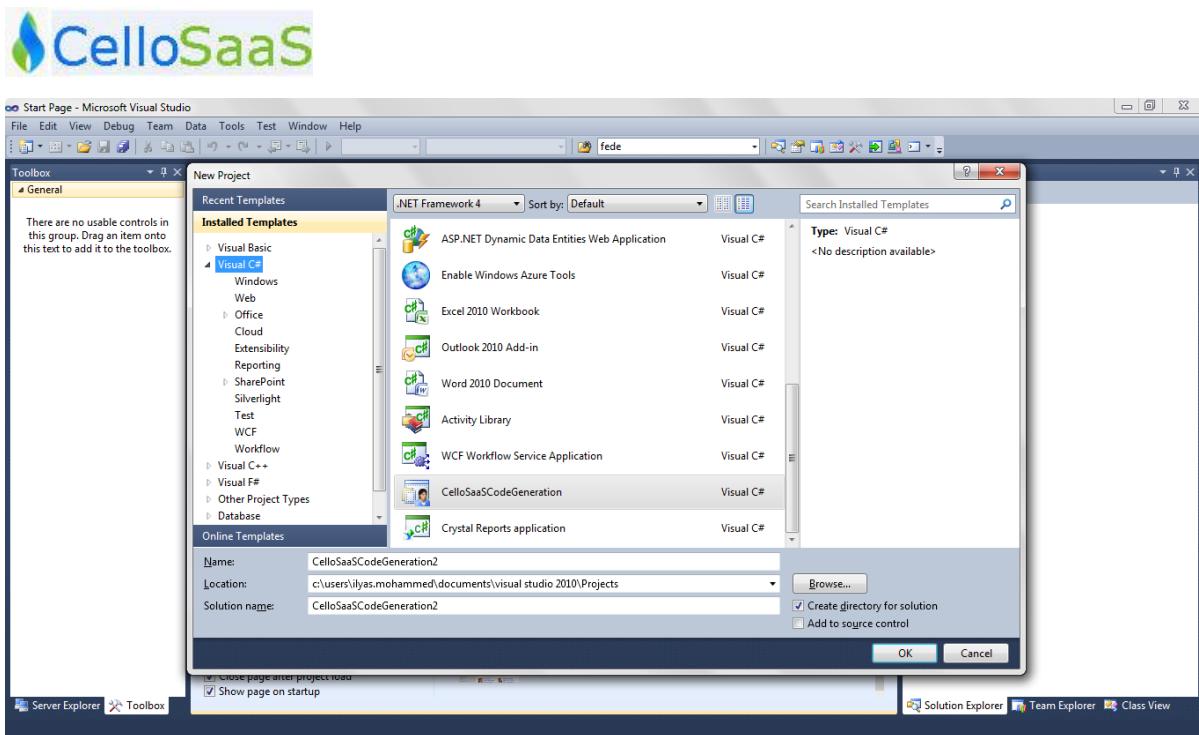


Figure 13-1 – New Project Screen

Code Generator

Code Generator is Productivity tool based on Text Templates, the input for the Code generator is tables in the database, it reads the structure of the table and creates all the necessary C# code for all the layers of the solution, For Ex, it creates the Data Access Layer, with the default DDL and DML scripts to carry out CRUD Operations. These templates are designed to emit code based on fixed design, the code generated follows certain guidelines, best practices etc.

Developers can then update the code created by code generator as per their requirement. The text templates shipped with CelloSaaS can also be modified to support specific needs of the developer. The Code generator project contains the below items apart from Text Templates. They are

- SourceCode Folder
- CodeGeneration.celloDSL

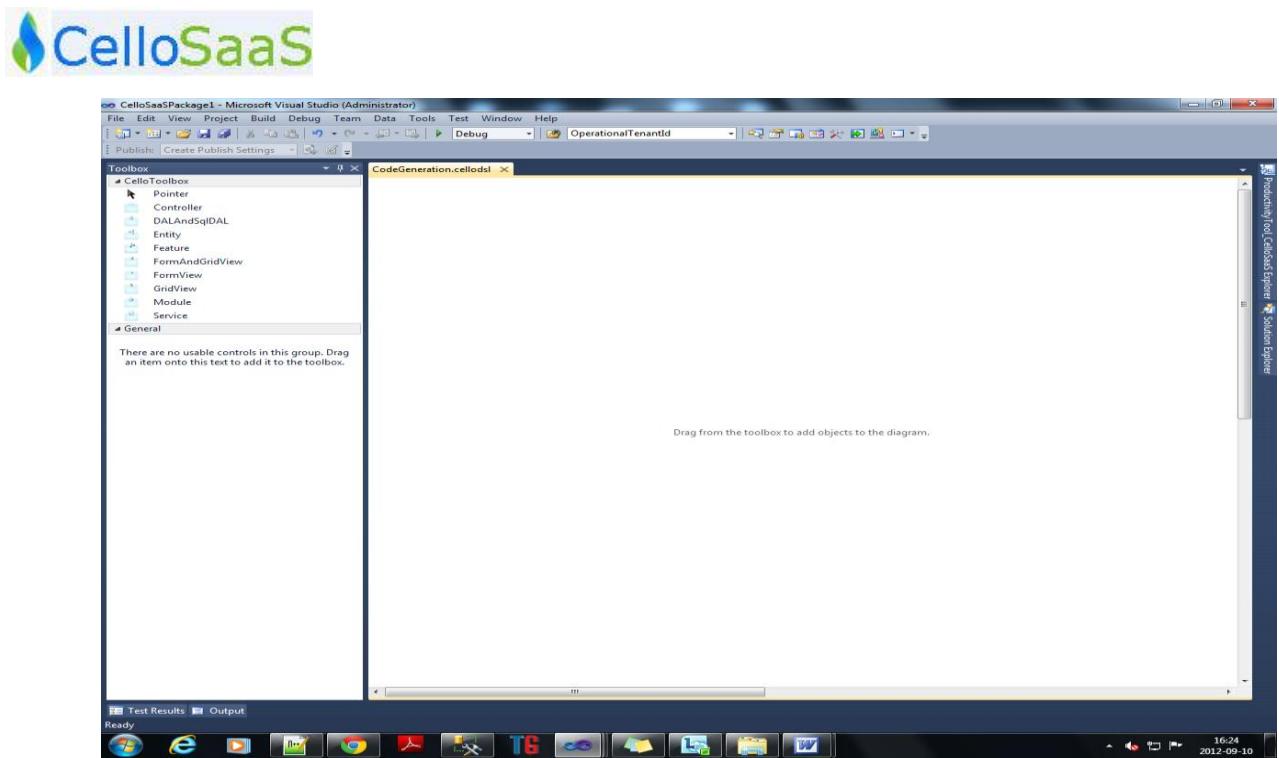


Figure 13-2 – New Project Screen

Source Code Folder

It's the destination folder in which the generated code will be stored. If you would like to save it in a different folder, you can do so by setting the application path setting in the **CodeGeneration.tt** Designer.

The designer is just like a form designer in Visual Studio, the tool box contains all the components to generate c# code. CelloSaaS framework follows certain implementation Pattern which has to be followed while writing the code. Below is the pictorial representation of the pattern.

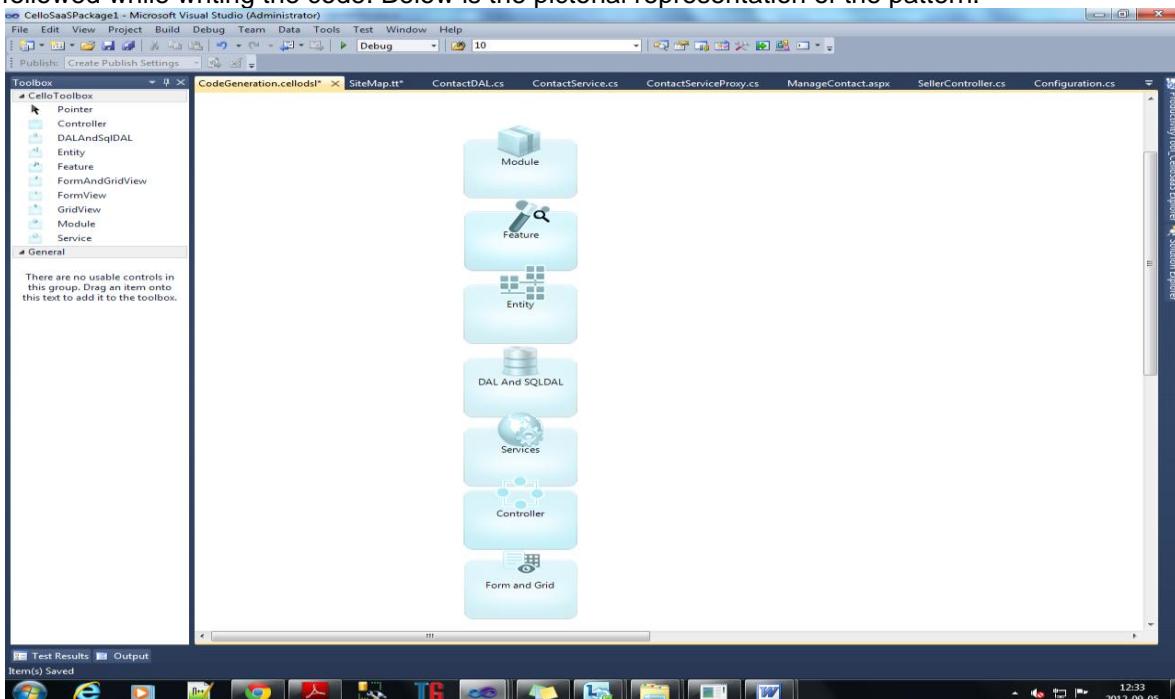


Figure 13-3 – New Project Screen

Components of Code Generator Module

Feature
 Entity
 DAL
 Service Contract
 Controller
 Form View
 Grid View
 Form and Grid View

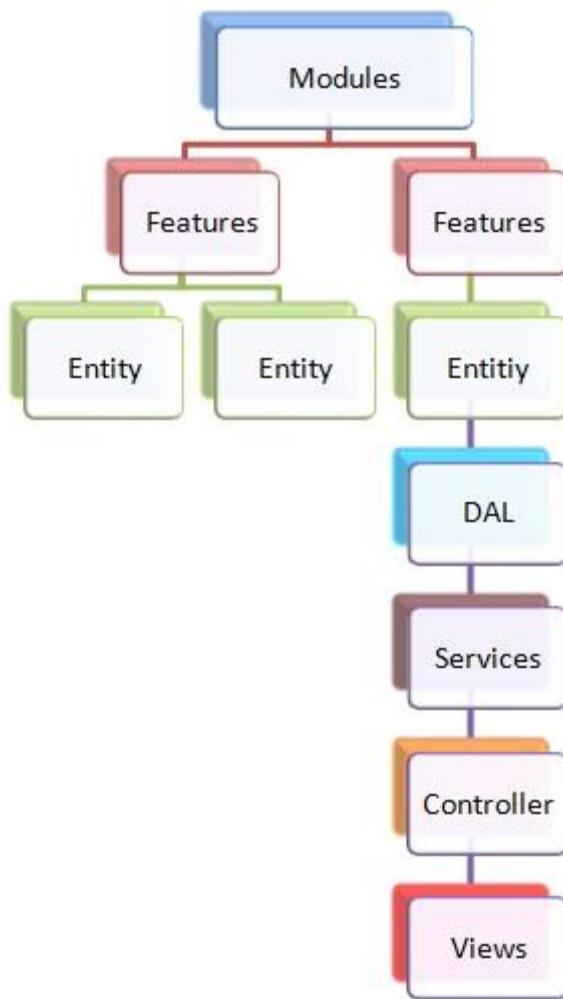


Figure 13-4 – Code Generator Components

Module

In the Context of CelloSaaS, group of entities will be collectively called as **Feature** and group of related and unrelated features are called as **module**. Module is the starting point of the code generator; feature or any other element cannot be added directly into the designer canvas without the module element. To begin with, the module object has to be dragged on to the Design Canvas followed by features and entities. Module name, feature name and entity name should not be same name. For example (entity name + Module)

Properties

- Connection String
- Name
- Package Name

It creates the module specific configuration creation including contents, Entity Permissions and sitemap etc. The core components which deal with FLUENT API related classes also get generated as part of the module element.

Feature

It is a collection of related tables or entities. Further, a feature element cannot contain an entity or other components directly and it has to be followed by a **feature** element. To add a new feature, select or highlight the **module** element before adding a feature under it. Again, a feature cannot host sub feature under but it can be hosted under the parent module. For example (entity name + Feature)

Properties

Name

Entity

An **entity** is a SQL table which contains one or more **fields**, **relations**, **constraints** and so on. A group of related or unrelated entities can be clubbed under a feature. For example entity name

Properties

Connection String

Name

Table Name

Other Privileges

Connection String Name

Extensible

Context Name [EDMX context Name]

EF DAL [True/False, True – Entity Framework DAL , False – ADO.Net DAL (Default)]

The **Entity** element generates the model class by observing the table's structure from the database by using the connection, it is critical to provide an error free connection string while configuring the parameters.

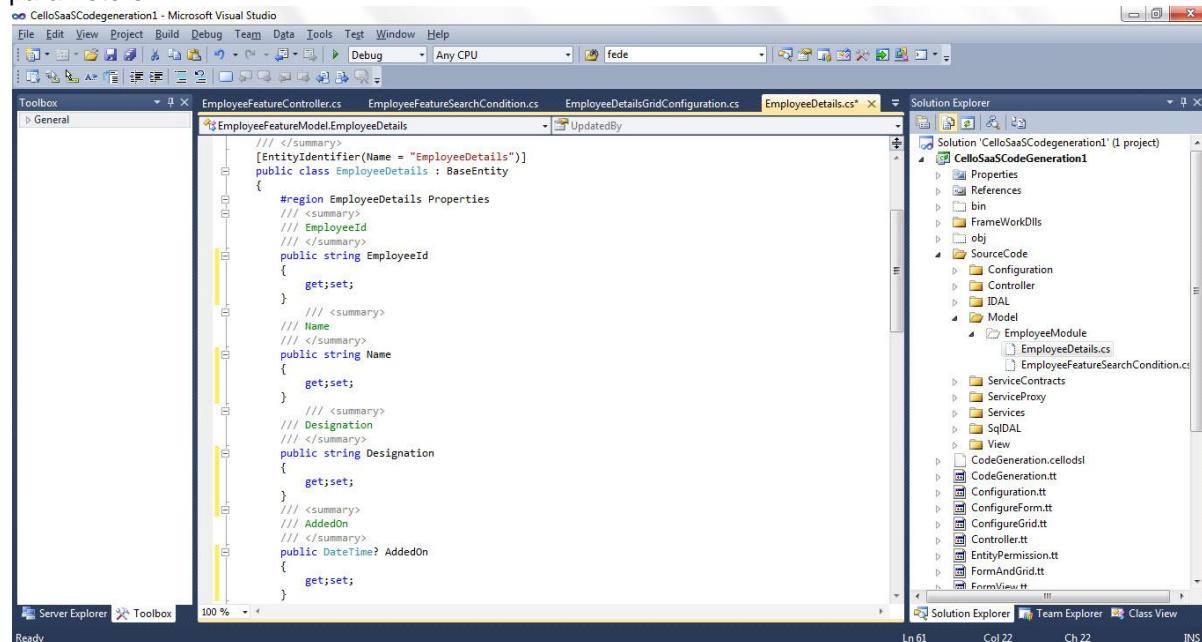


Figure 13-5 – Module Screen

DAL & SQLDAL

The DAL & SQLDAL element takes care of creating class files which provides the MS SQL query implementation for all the default functionalities such as Select, Update and Delete. The SQL DAL uses ad-hoc query building methodology, which eases the complication of customizing the base

query which is issued to the Database server. The developer can build or construct the custom/dynamic query and attach with the base query during the run time. The naming conventions are taken from the properties that are assigned while creating the objects at the design time. The methods are decorated with the necessary comments which will help the developers to maintain the code with ease. The class will provide the necessary method definition while implementing the interface. For example (I + entity name + DAL, entity name + DAL)

Properties

Name
SQL DAL Name

Service

The Service element generates class file which provides default definition for the methods defined in the service contract. In case if the developer wants to modify the default definition, he/she can then very well go-ahead and modify the same according to their requirement. The definition includes the privilege context which takes care of data isolation and the privilege mapping as per the roles and the responsibilities mapping which can be assigned later by the product Administrator. For example (I + entity name + Service, entity name + Service, entity name + ServiceProxy)

Properties

Service Contracts Name
Service Name
Service Proxy Name

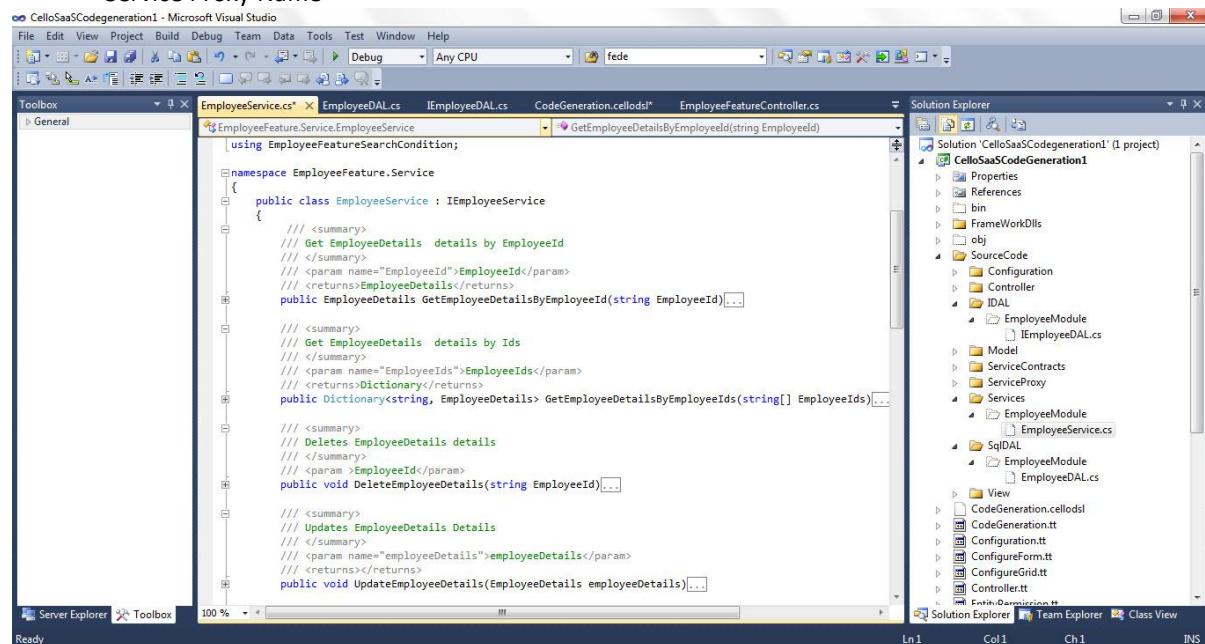


Figure 13-6 – Service Screen

Service Proxy

Service proxy, as the name defines it creates definition methods which internally connect the relevant implementations, because the framework is designed to provide high integrity, security, manageability etc. So it is advised not to expose the method which has the implementation rather use the proxy method while developing the business application on top of the framework as well. The developers are advised to follow the same design when writing class, interface structure in order to stick with the same design pattern as Techcello.

```

using System.Linq;
using CelloSaaS.Library;
using EmployeeFeature.Model;
using EmployeeFeature.ServiceContracts;
namespace EmployeeFeature.ServiceProxy
{
    public class EmployeeServiceProxy
    {
        #region EmployeeDetails

        /// <summary>
        /// Adding EmployeeDetails Details
        /// </summary>
        /// <param name="EmployeeDetails"></param>
        /// <returns>string</returns>
        public static string AddEmployeeDetails(EmployeeDetails employeeDetails){...}

        /// <summary>
        /// Updating EmployeeDetails Details
        /// </summary>
        /// <param name="employeeDetails"></param>
        public static void UpdateEmployeeDetails(EmployeeDetails employeeDetails){...}

        /// <summary>
        /// Deleting EmployeeDetails Details
        /// </summary>
        /// <param name="EmployeeId"></param>
        public static void DeleteEmployeeDetails(string EmployeeId){...}

        /// <summary>
        /// GetEmployeeDetails Details by employeeDetailsid
        /// </summary>
        /// <param name="EmployeeId"></param>
        /// <returns>EmployeeDetails</returns>
        public static EmployeeDetails GetEmployeeDetailsByEmployeeId(string EmployeeId){...}
    }
}

```

Figure 13-7 – Service Proxy Screen

Controller

The code generator will create the controller class with the necessary methods to manage the entity data. The controller name should not be suffixed with word ‘Controller’. The basic methods are follows

Properties

```

public class EmployeeFeatureController : Controller
{
    #region EmployeeDetails

    /// <summary>
    /// This class holds action for EmployeeFeature controller
    /// </summary>

    public ActionResult ManageEmployeeDetails(string EmployeeId){...}

    /// <summary>
    /// This method is to load the EmployeeDetails details to ManageEmployeeDetails page
    /// </summary>
    /// <param name="EmployeeId"></param>
    /// <returns>view</returns>
    public ActionResult ManageEmployeeDetails(string EmployeeId){...}

    /// <summary>
    /// This method Add and Update EmployeeDetails Details
    /// </summary>
    /// <param name="formCollection"></param>
    /// <returns>view</returns>
    [AcceptVerbs(HttpVerbs.Post)]
    public ActionResult ManageEmployeeDetails(FormCollection formCollection){...}

    /// <summary>
    /// Deleting EmployeeDetailsDetails
    /// </summary>
    /// <param name="EmployeeId"></param>
    /// <returns>view</returns>
    public ActionResult DeleteEmployeeDetails(string EmployeeId){...}

    /// <summary>
    /// Checking validations to employeeDetails
    /// </summary>

```

Figure 13-8 – Controller Screen

Later, the developer can add business logic and customize the default methods based on their requirement. The methods and members also decorated with necessary attributes and verbs which will make the methods Restful and JSON Serialized.

Form View

It provides the basic HTML view to add, update and delete the data of the entity. The form will be designed automatically based on the number fields and data types. Configure the fields that you would like to display in the view. Currently, the validation support has not been provided by the framework, but the developers can add custom validations for all the fields or specific fields.

Properties

Name

Grid View

Grid view provides the list view facility for the entity's grid based data. Configure the fields that you would like to display in the view. It comes with the default sorting feature enabled along with the pagination support and there is no absolute restriction on modifying the look and feel of the Grid. Internally, it uses Cello Grid component to emit the required html to create the table.

Properties

Name

Form and Grid View

This is the combination of both Form and grid view. Configure the fields that you would like to display in the view. In most cases the developer might need to display content from the database as well as manipulate them on the same screen. So this provides necessary forms to add, edit and delete the data from the entity and grid view facilitates by rendering the data in the grid format and it can also be completely controlled by the developer.

Properties

Form Name

Grid Name

13.1.4 Steps to follow to start generating the code

Sample

This is a step by step guide to create the class files for **Employee Entity** with the default CelloSaaS structure to understand how to work with Code Generator. Prior to generating code, make sure the database and table are exists.

- ☞ **Step1:** Open **CodeGeneration.tt** and Set the ApplicationPath/SourceCode in CodeGeneration.tt and Package name as your solution name.
- ☞ Double click or Drag and Drop the Module element into the designer canvas and name it. The name you provide will be taken for all the naming conventions, so provide related/appropriate naming conventions.

Module Name, Feature Name and Entity Name should not be the same

Properties

Connection String: Data Source=servername;Initial Catalog=dbname;User Id=sa;Password=*****;

Module Name: EmployeeManagementSystem

Package Name: CelloSaaSPackage1

- ☞ **Step2:** Highlight>Select module element, drag and drop the Feature element and fill the required parameters as discussed above

Properties

FeatureName: ManageEmployee

- ☞ **Step3:** Highlight the Feature Element, drag and drop the Entity element and fill in the required parameters.

Properties

Connection String: Data Source=servername;Initial Catalog=dbname;User Id=sa;Password=*****;

Connection String Name(Optional): <ModuleName> ConnectionString ,but connection string name already have property value as ApplicationconnectionString

Entity Name :<EntityName>

Extensible (Optional): <True/False> Note: Based on this property, the model generated will be decorated with appropriate Attributes, set true if the entity needs to be extendible.

Other Privileges (Optional): SearchEmployee, GroupEmployee etc

Table Name: <Table Name>



This refers to the actual table name in the database, so make sure the name given matches with the table in the database. Make sure, the connection string property has no issues, failing Code generator might throw error as not able to pull the table/entity details from the database.

If you set the connection string property in module then the entity will use the same connection else each entity will use the connection string property provided.

- ☞ **Step4:** Highlight Entity element, drag and drop the **DAL&SQLDAL** element into the design surface and fill in the required parameters.

Properties

DAL Name: I<EntityName>DAL



This is an interface, so prefix with I

SQL DALName: <EntityName>DAL

- ☞ **Step5:** Highlight **DAL&SQLDAL** element, drag and drop the **Service** element into the design surface and fill in the required parameters.

Properties

Service Contracts Name: I< EntityName >Service

Service Name: < EntityName >Service

Service Proxy Name: < EntityName >ServiceProxy

- ☞ **Step6:** Highlight **Service** element, drag and drop the **Controller** element into the design surface and fill in the required parameters.

Properties

Controller Name: <EntityName>

- ☞ **Step7:** Highlight **Controller** element, drag and drop the **Form/Grid/Form&Grid View** element into the design surface and fill in the required parameters.

The word “Controller” should not be suffixed in controller name.

Properties

FormName: <FormName>

GridName: <GridName>

Tip: The Code Generator has been designed in such a way that, the elements can be manipulated only in the above order, means the developer cannot use any other elements under module other than Feature element. Similarly the condition has been imposed, so developers need not worry about the order of placing the elements one after another.

- ☞ **Step8:** After placing the respective Tools and configuring the Parameters. Click on → Transform Template icon → Solution Explorer

The Code generator will generate the necessary code in the Source Code Folder. Please follow the below step to move these files to respective application Project Folder.



Once the File is copied to the destination project, make sure the file is included as part of the project and immediately compile the project and add the necessary references and make the project has no compile time errors.



Entity Name/Package name/Module name should not be the same

Step By Step Process to Place the Files in the Destination Application

- ☞ Right click on the SourceCode folder and select the "Open in Windows Explorer"

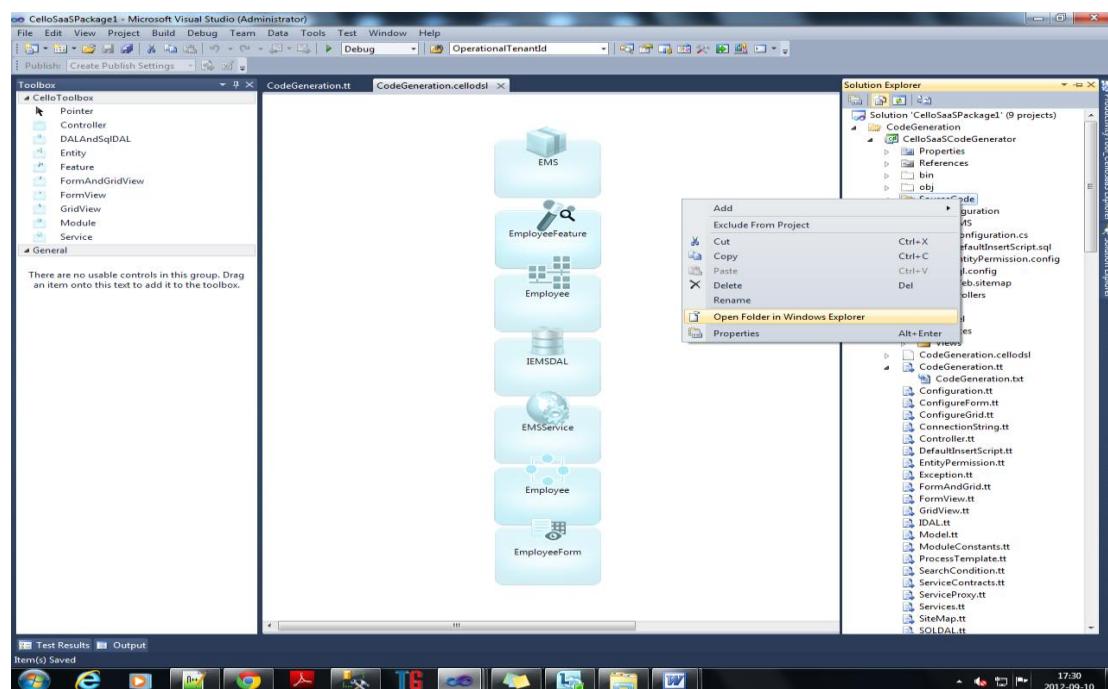


Figure 13-9 – Gode Generator Screen

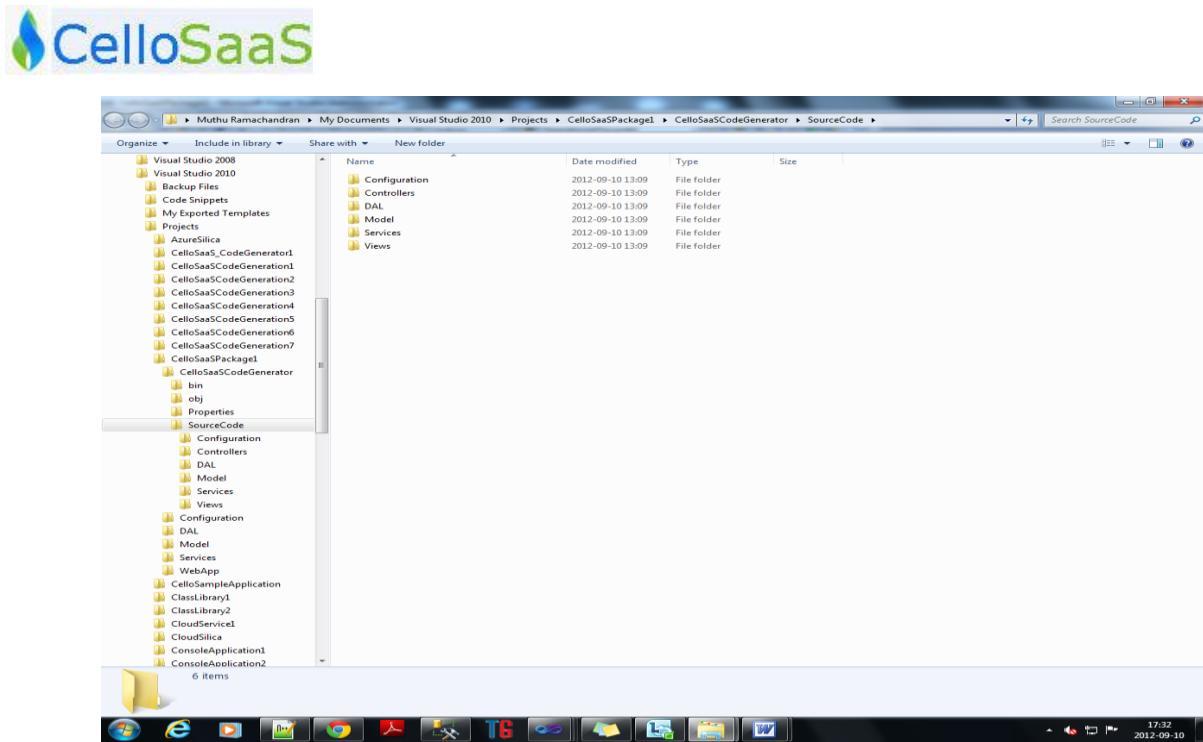


Figure 13-10 – Windows Explorer Screen

→ Right click on the Solution and select the "Open in Windows Explorer"

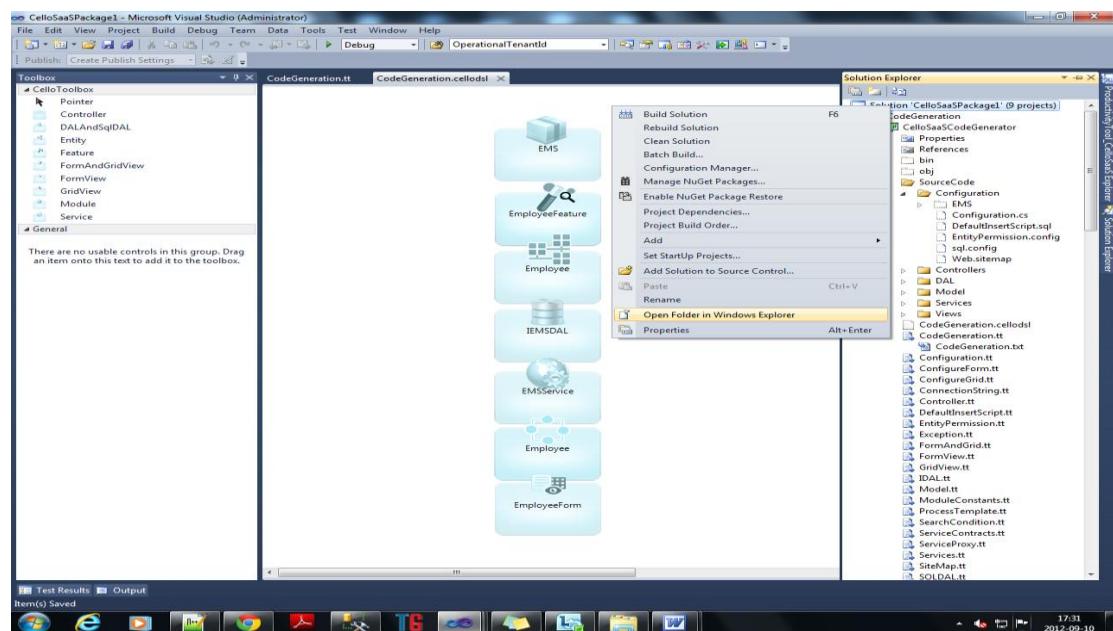


Figure 13-11 – Gode Generator Screen

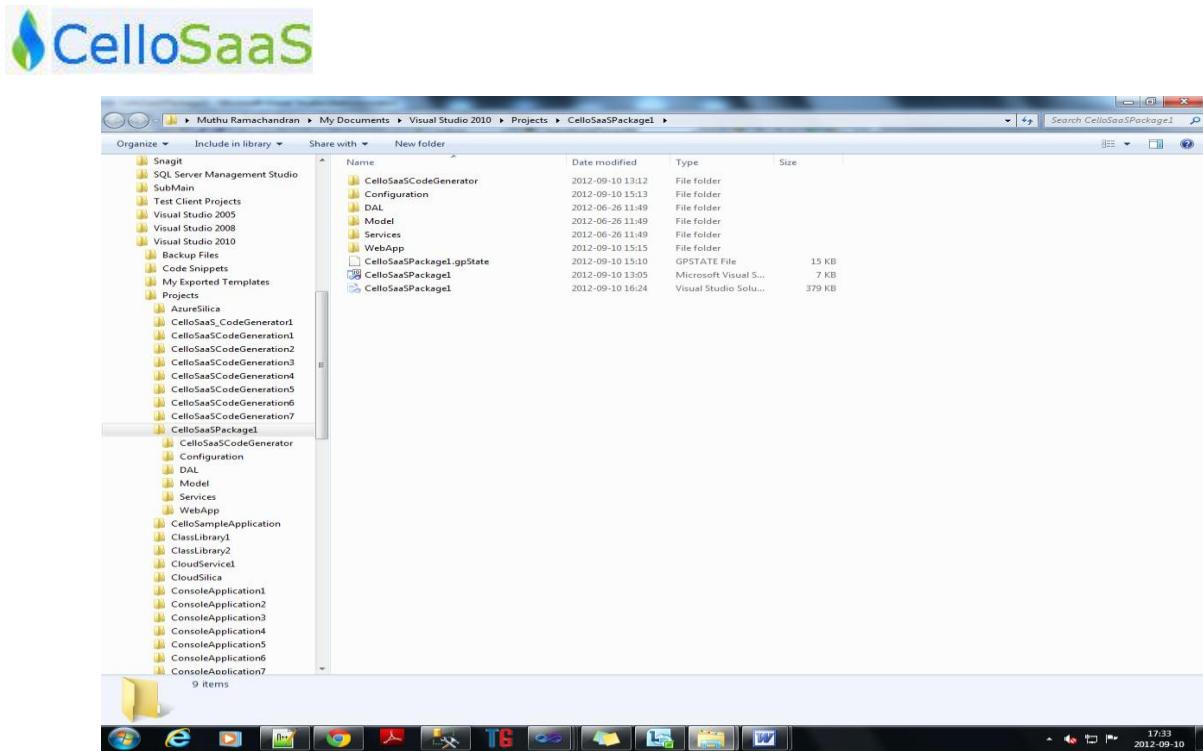


Figure 13-12 - Windows Explorer Screen

- ☞ Copy and paste the Model and DAL and Service and Configuration from Source code folder to the Solution folder.

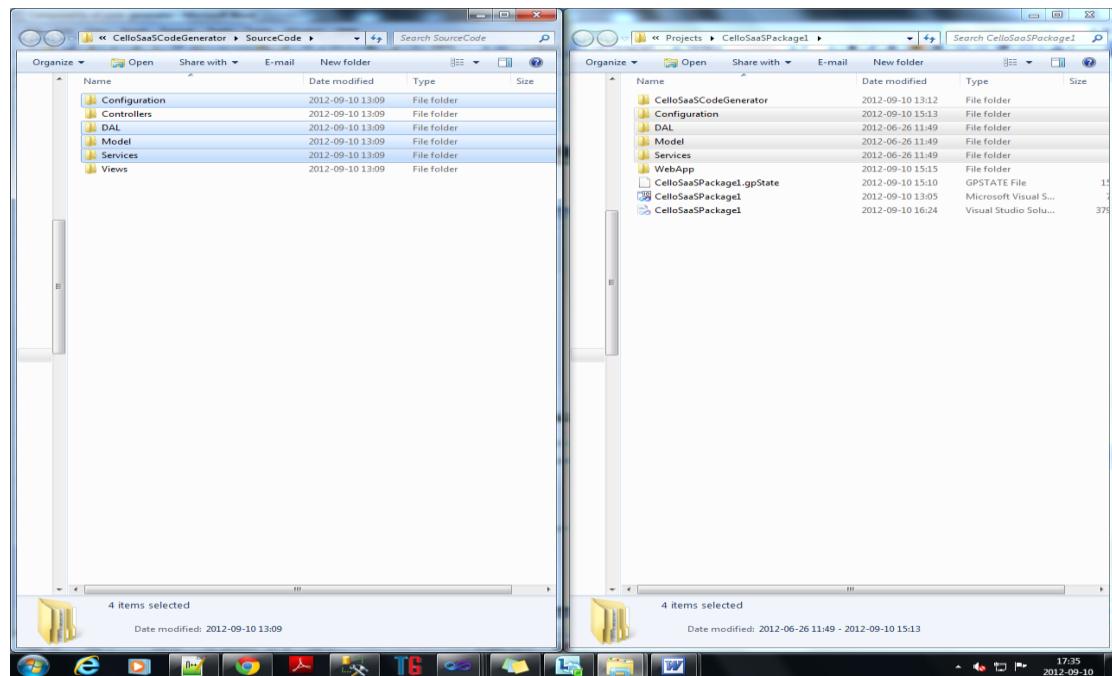


Figure 13-13 – Model and DAL Service Configuration Screen

- ☞ Then remove the Web.Sitemap, Sql.config, DefaultInsertScript.sql, and EntityPermission.config from the Configuration folder.

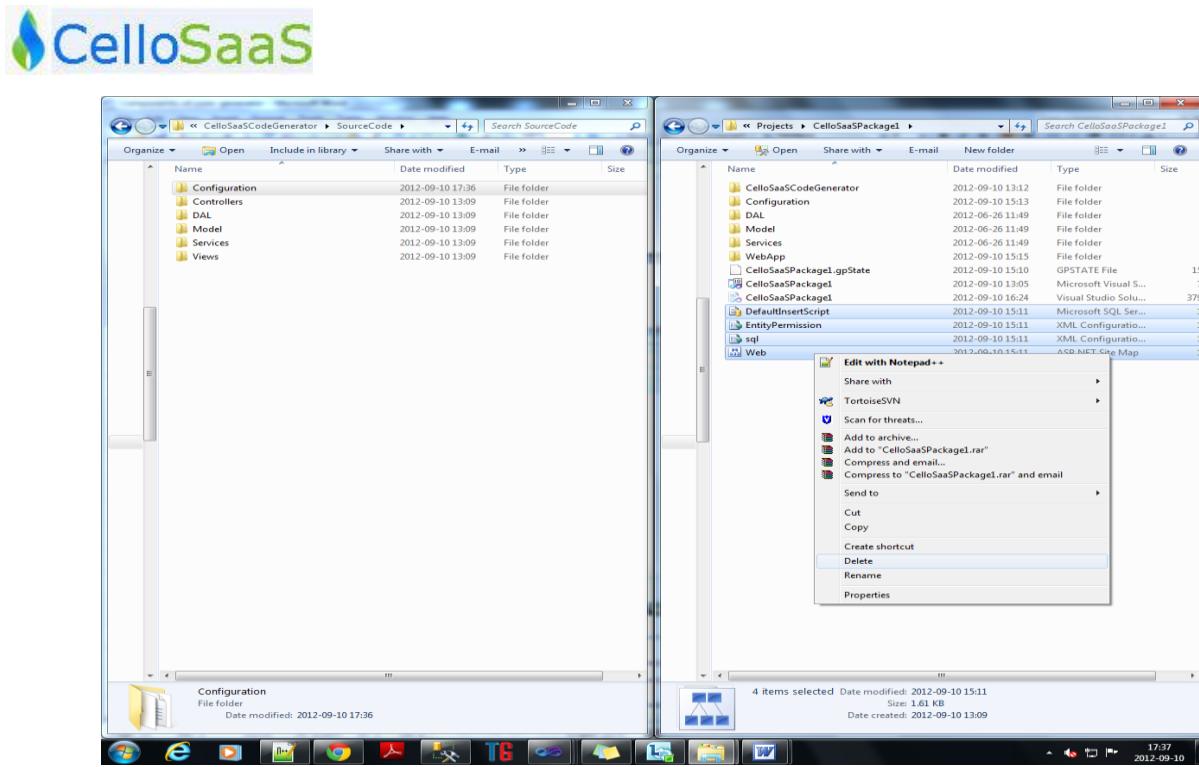


Figure 13-14 - Model and DAL Service Configuration Screen

☞ Copy the Controllers and Views folder from Source code folder to Web Application folder

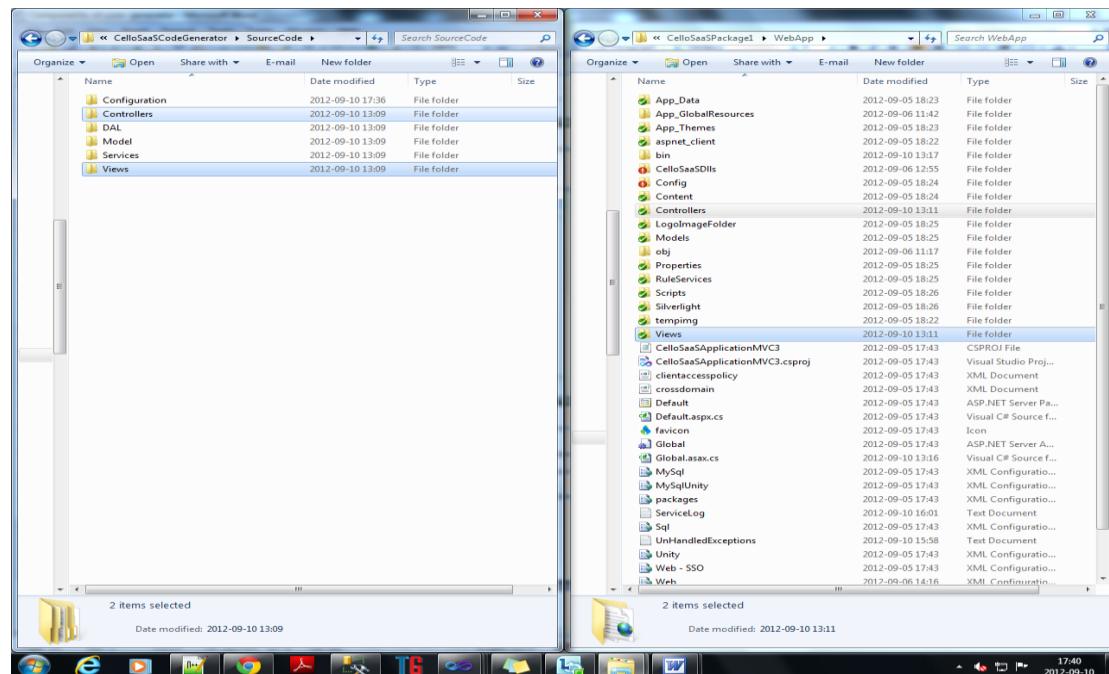


Figure 13-15 - Model and DAL Service Configuration Screen

☞ Copy the content of the file(Web.Sitemap, sql.config and EntityPermission.config) to paste it in appropriate files(Entitypermission file is available in Config folder inside Web Application Folder)

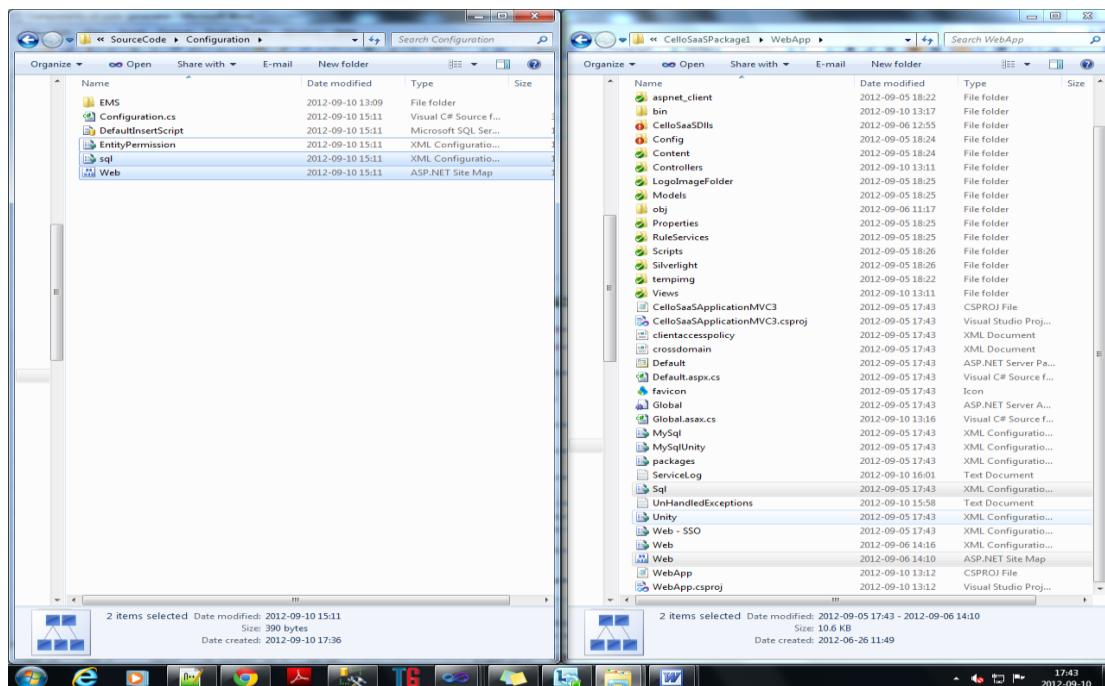


Figure 13-16 -- Model and DAL Service Configuration Screen

☞ References need to be added in respective project files.

Configuration files :

Create a solution folder and name it as Configuration. Create a C# class library project [remove the default class1.cs]. Place the generated configuration file(Configuration,formconfiguration,grid configuration)inside this project.

Finally make sure, the respective projects references the following libraries properly.

Project File	Assembly
Model	None
Configuration	Model,CelloSaaS.Configuration,CelloSaaS.Model,CelloSaaS.Library
IDAL	Model
SQLDAL	Model,IDLAL
ServiceContracts	Model
Services	Model,DAI,ServiceContracts,SQLDAL
ServiceProxy	Model,ServiceContracts,Services,SQLDAL
WepApp	Model,ServiceProxy,ServiceContracts

☞ Run the Default Insert scripts in to the database



In CelloSaaS framework, we would add module and feature to the application, so that module and feature information should be available in unrestricted package. You can then only access the module and feature in Product admin login and also to assign the child tenant.

 Add the following code in **global.asax** file [Purpose]

Here we are adding the code to initialize the fluent api configuration then only you can access form, entity and everything.

CelloSaaS Package1 - Microsoft Visual Studio (Administrator)

File Edit View Refactor Project Build Debug Team Data Tools Test Window Help

OperationalTenantId

GlobalAsax.cs CodeGeneration.tt CodeGeneration.cellodsl

General

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
private void Configure()
{
    // Register the rules
    CelloSaaS.Configuration.CelloConfigurator.RegisterModule<CelloSaaS.Notification.NotificationModuleConfigurator>();
    CelloSaaS.Configuration.CelloConfigurator.RegisterModule<CelloSaaS.WorkFlow.WorkFlowModuleConfigurator>();
    CelloSaaS.Configuration.CelloConfigurator.RegisterModule<CelloSaaS.DataBackup.DataBackupModuleConfigurator>();

    // Register the entities
    CelloSaaS.Configuration.CelloConfigurator.RegisterEntity<CelloSaaS.WorkFlow.WorkFlowEntityConfigurator>();

    CelloSaaS.Configuration.CelloConfigurator.RegisterModule<CelloSaaS.Configuration.DBCellobModuleConfigurator>();
    CelloSaaS.Configuration.CelloConfigurator.RegisterEntity<CelloSaaS.Configuration.DBCellobEntityConfigurator>();
    CelloSaaS.Configuration.CelloConfigurator.RegisterDataView<CelloSaaS.Configuration.DBCellobEntityViewConfigurator>();

    CelloSaaS.Configuration.CelloConfigurator.RegisterModule<EMS.Configuration.EMSConfiguration.EMSConfigurator>();
    CelloSaaS.Configuration.CelloConfigurator.RegisterEntity<EMS.Configuration.EMSConfiguration.EMSEntityConfigurator>();
    CelloSaaS.Configuration.CelloConfigurator.RegisterDataView<EMS.Configuration.EMSConfiguration.EMSDataviewConfigurator>();
    CelloSaaS.Configuration.CelloConfigurator.Configure();
}

private void InitRules()
{
    RuleConfiguration.EntityRules.Add<CelloSaaS.Model.UserManagement.UserDetails>()
        .WithPreprocessorRule()
        .WithValidationRule()
        .Add<CelloSaaS.Model.TenantManagement.Address>()
        .WithPreprocessorRule()
        .WithValidationRule()
        .Add<CelloSaaS.Model.TenantManagement.TenantDetails>()
        .WithValidationRule()
        .WithPreprocessorRule();

    Rules.Initialise("CelloSaaConnectionString");
}

private void RegisterBinders()
{
    ModelBinders.Binders.DefaultBinder = new CelloSaaS.View.BusinessModelBinder();
    ModelBinders.Binders.Add(typeof(CelloSaaS.Model.DataManagement.ExtendedEntityRow), new CelloSaaS.View.ExtendedEntityB
}
```

Test Results Output

Ready

Ln 86 Col 117 Ch 117

100 %

17:45 2012-09-10

```
CelloSaaS.Productivity.CelloConfigurator.RegisterModule<ModuleConfigurator>();  
CelloSaaS.Productivity.CelloConfigurator.RegisterEntity<EntityConfigurator>();  
CelloSaaS.Productivity.CelloConfigurator.Register DataView< DataViewConfigurator>();
```

Then include the entire file into the solution and compile the Application and make sure all the class files compiled successfully without throwing any compile time errors.

Extended Table Structure

There are many ways of implementing Extended Fields capability. They are

- Key-Value Pair Model

- Flat Table Model

- Key-Value Pair Model

Below is the create table script which contains the place holder to be replaced with the entity table name.

Example: If the Entity Name is Customer then the following script has to be executed

Table 1: Table to Store Extended Field Properties

```

CREATE TABLE [dbo].[<TableName>Extn](
[CustomerExtn_Id] [uniqueidentifier] NOT NULL,
[CustomerExtn_EntityId] [varchar](200) NOT NULL,
[CustomerExtn_Refrenceld] [uniqueidentifier] NOT NULL,
[CustomerExtn_TenantId] [uniqueidentifier] NOT NULL,
[CustomerExtn_CreatedOn] [datetime] NOT NULL,
[CustomerExtn_CreatedBy] [varchar](50) NOT NULL,
[CustomerExtn_UpdatedOn] [datetime] NULL,
[CustomerExtn_UpdatedBy] [varchar](50) NULL,
[CustomerExtn_Status] [bit] NOT NULL,
CONSTRAINT [PK_CustomerExtn] PRIMARY KEY CLUSTERED
(
[CustomerExtn_Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_Entity] FOREIGN KEY([CustomerExtn_EntityId])
REFERENCES [dbo].[Entity] ([Entity_ID])
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_Entity]
ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_Customer] FOREIGN KEY([CustomerExtn_Refrenceld])
REFERENCES [dbo].[Customer] ([Tenant_Code])
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_Customer]
ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_Customer1] FOREIGN KEY([CustomerExtn_TenantId])
REFERENCES [dbo].[Customer] ([Tenant_Code])
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_Customer1]
ALTER TABLE [dbo].[CustomerExtn] ADD CONSTRAINT [DF_CustomerExtn_CustomerExtn_Id]
DEFAULT (newsequentialid()) FOR [CustomerExtn_Id]

```

Table 2: Table to Store Extended Field values

```

CREATE TABLE [dbo].[CustomerExtn](
[CustomerExtn_Id] [uniqueidentifier] NOT NULL,
[CustomerExtn_EntityId] [varchar](200) NOT NULL,
[CustomerExtn_Refrenceld] [uniqueidentifier] NOT NULL,
[CustomerExtn_TenantId] [uniqueidentifier] NOT NULL,
[CustomerExtn_CreatedOn] [datetime] NOT NULL,
[CustomerExtn_CreatedBy] [varchar](50) NOT NULL,
[CustomerExtn_UpdatedOn] [datetime] NULL,
[CustomerExtn_UpdatedBy] [varchar](50) NULL,

```

```

[CustomerExtn_Status] [bit] NOT NULL,
CONSTRAINT [PK_CustomerExtn] PRIMARY KEY CLUSTERED
(
[CustomerExtn_Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_Entity] FOREIGN KEY([CustomerExtn_EntityId])
REFERENCES [dbo].[Entity] ([Entity_ID])
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_Entity]
ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_Customer] FOREIGN KEY([CustomerExtn_Refrenceld])
REFERENCES [dbo].[Customer] ([Tenant_Code])
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_Customer]
ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_Customer1] FOREIGN KEY([CustomerExtn_TenantId])
REFERENCES [dbo].[Customer] ([Tenant_Code])
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_Customer1]
ALTER TABLE [dbo].[CustomerExtn] ADD CONSTRAINT [DF_CustomerExtn_CustomerExtn_Id]
DEFAULT (newsequentialid()) FOR [CustomerExtn_Id]

```

Flat Table

The following script is flat table create script, you have to replace your entity instead of customer. If you are using new extended field Flat Table technique you have to follow this way.

```

***** Object: Table [dbo].[CustomerExtn]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[CustomerExtn](
[CustomerExtn_Id] [uniqueidentifier] NOT NULL,
[CustomerExtn_EntityId] [varchar](200) NOT NULL,
[CustomerExtn_Refrenceld] [uniqueidentifier] NOT NULL,
[CustomerExtn_TenantId] [uniqueidentifier] NOT NULL,
[CustomerExtn_Column1] [int] NULL,
[CustomerExtn_Column2] [int] NULL,
[CustomerExtn_Column3] [int] NULL,
[CustomerExtn_Column4] [int] NULL,
[CustomerExtn_Column5] [int] NULL,
[CustomerExtn_Column6] [datetime] NULL,
[CustomerExtn_Column7] [datetime] NULL,
[CustomerExtn_Column8] [datetime] NULL,
[CustomerExtn_Column9] [datetime] NULL,
[CustomerExtn_Column10] [datetime] NULL,
[CustomerExtn_Column11] [varchar](50) NULL,
[CustomerExtn_Column12] [varchar](50) NULL,
[CustomerExtn_Column13] [varchar](50) NULL,
[CustomerExtn_Column14] [varchar](50) NULL,
[CustomerExtn_Column15] [varchar](50) NULL,
[CustomerExtn_Column16] [varchar](max) NULL,
[CustomerExtn_Column17] [varchar](max) NULL,
[CustomerExtn_Column18] [varchar](max) NULL,
[CustomerExtn_Column19] [varchar](max) NULL,
[CustomerExtn_Column20] [varchar](max) NULL,

```

```

[CustomerExtn_Column21] [float] NULL,
[CustomerExtn_Column22] [float] NULL,
[CustomerExtn_Column23] [float] NULL,
[CustomerExtn_Column24] [float] NULL,
[CustomerExtn_Column25] [float] NULL,
[CustomerExtn_Column26] [bit] NULL,
[CustomerExtn_Column27] [bit] NULL,
[CustomerExtn_Column28] [bit] NULL,
[CustomerExtn_Column29] [bit] NULL,
[CustomerExtn_Column30] [bit] NULL,
[CustomerExtn_CreatedOn] [datetime] NOT NULL,
[CustomerExtn_CreatedBy] [varchar](50) NOT NULL,
[CustomerExtnUpdatedOn] [datetime] NULL,
[CustomerExtnUpdatedBy] [varchar](50) NULL,
[CustomerExtn_Status] [bit] NOT NULL,
CONSTRAINT [PK_CustomerExtn1] PRIMARY KEY CLUSTERED
(
    [CustomerExtn_Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_Customer1] FOREIGN KEY([CustomerExtn_ReferecId])
REFERENCES [dbo].[Customer] ([Customer_ID])
GO
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_Customer1]
GO
ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_Entity2] FOREIGN KEY([CustomerExtn_EntityId])
REFERENCES [dbo].[Entity] ([Entity_ID])
GO
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_Entity2]
GO
ALTER TABLE [dbo].[CustomerExtn] WITH CHECK ADD CONSTRAINT
[FK_CustomerExtn_TenantDetails1] FOREIGN KEY([CustomerExtn_TenantId])
REFERENCES [dbo].[TenantDetails] ([Tenant_Code])
GO
ALTER TABLE [dbo].[CustomerExtn] CHECK CONSTRAINT [FK_CustomerExtn_TenantDetails1]
GO
ALTER TABLE [dbo].[CustomerExtn] ADD CONSTRAINT [DF_CustomerExtn_CustomerExtn_Id1]
DEFAULT (newsequentialid()) FOR [CustomerExtn_Id]
GO

```

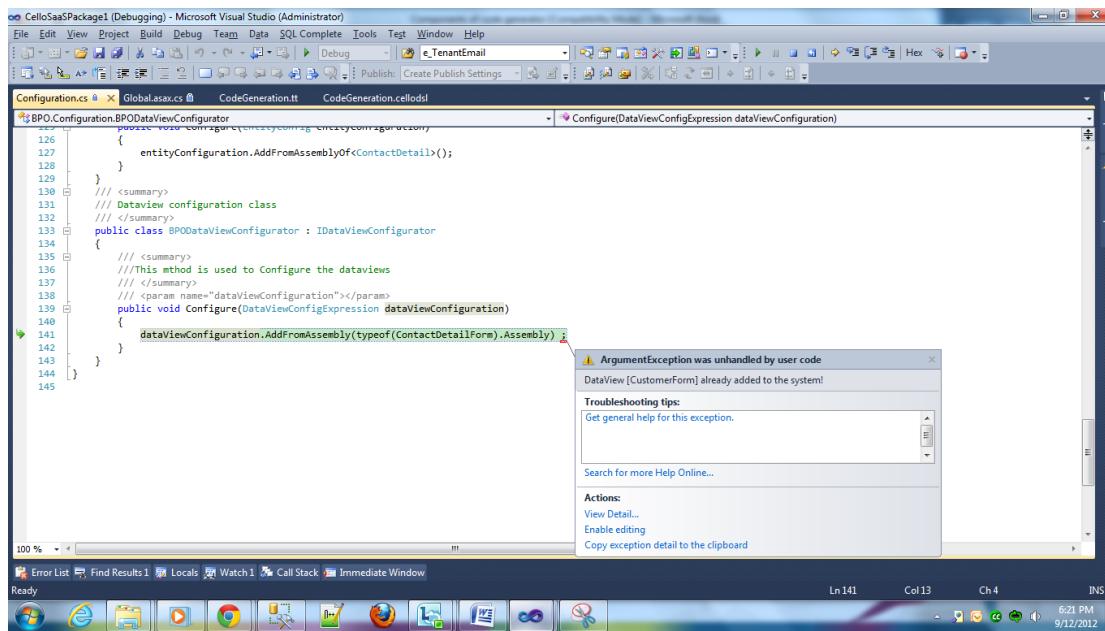
Possible Errors & Solutions

Error 1:

DataView [FormName] already added to the system.

Solution:

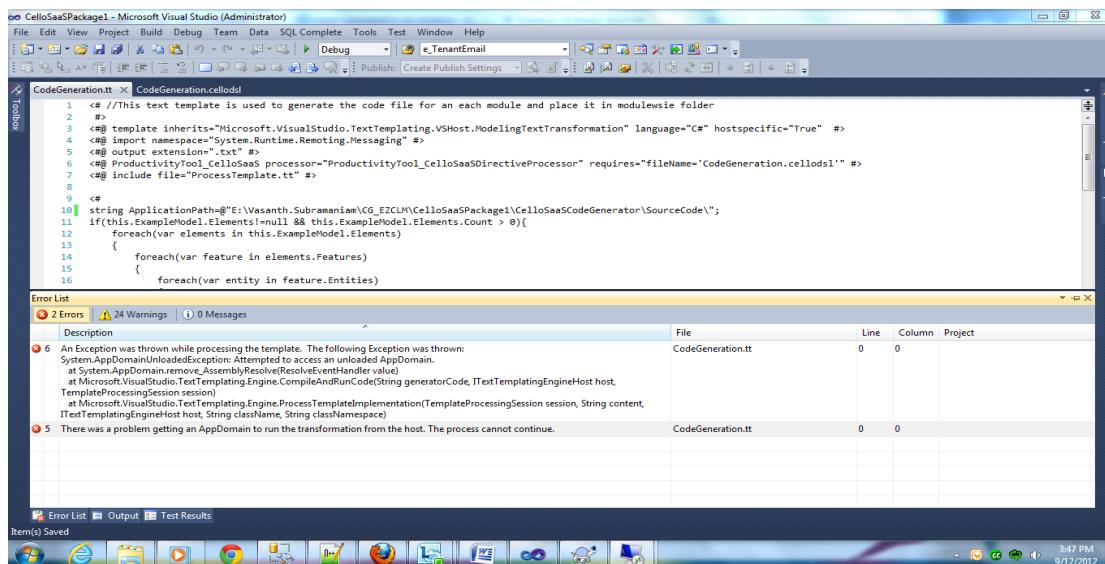
Kindly check the form id and grid id of the data view whether the id is repeating in any one of the form kindly change it properly.



Error 2:

AppDomain Problem

When you click on transform all template icon, you might get the below error.



Solution

This error occurs due to AppDomain problem so restart your visual studio then open your solution the problem will be resolved.

Advantages of Code Generator

Sleek Interface

CelloSaaS brings you powerful features in an easy to use interface. Everything you need to generate thousands of lines of ASP.NET code in seconds - at a click of a button.



Proven Architecture, based on Microsoft Best Practices

The code generated adheres to Microsoft Best Practices as well as CelloSaaS specific coding standard and architecture that lead to high quality light weight code. Code Generator emits well commented code that is highly customizable and gives the developer the ultimate control over the generated code.

- Supports Visual Studio 2010
- Each should contain atleast one primary key
- ID Should be UniqueIdentifier [NewSequentialID()]
- TenantId[TenantID is must]
- Audit Fields are optional

Scope of Code Generator

Cello Code generator is not a RAD [Rapid Application Development] to generate code without the help of developers; rather it is a tool to generate C# code as per Cello Stanadards and best practices for all the layers of the web application.

Using Code generator User can write code only for single entity. It cannot generate code for entities with referential relationship

Code generator will emit code only for a single entity and the reference relationships to the entity need to be managed by the developers manually.

Note:

The Username that is mentioned in the connection string must have sufficient permission to access the objects, in order to fetch the complete details about the object.

The regular/base field names should not contain any of the below audit field naming conventions

Audit Field naming Convention

["UpdatedBy","EditedBy","UpdatedOn","EditedOn","CreatedBy","AddedBy","CreatedOn","AddedOn","Status"]

13.2 Utilities

From time to time, CelloSaaS may come up with different utilities. These utilities may not be a part of CelloSaaS solution itself – but, may be designed and developed for some specific purposes. For example, database maintenance, view generation, etc

14 WINDOWS COMMUNICATION FOUNDATION (WCF) SERVICE

CelloSaaS service methods are exposed as web services using windows communication foundation (WCF). It uses transport and message level security mode which ensures proper security for the data being transmitted.

This topic outlines the basic steps required to consume a Windows Communication Foundation (WCF) service that is hosted in Internet Information Services (IIS).

Deploying CelloSaaS WCF service

- ☞ Create a certificate using Makecert or Selfcert (For example, Localhost.cer).
- ☞ Save **CelloSaaS certificate** into your local system.
- ☞ Go to **Start > Run**. Type **mmc**. Click **Ok** button. The Console 1 screen gets displayed.

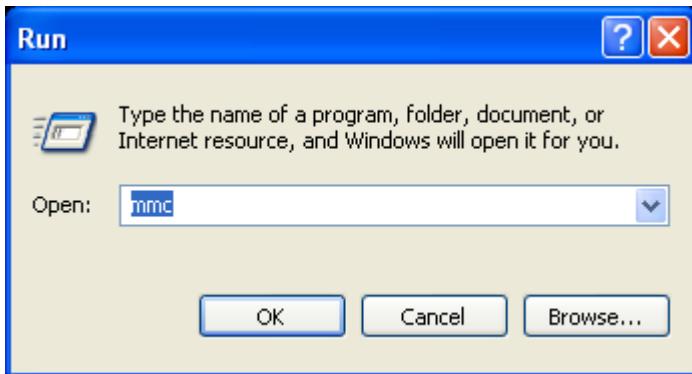


Figure 14-1 – Run Command

- ☞ Click File>Add/Remove Snap-in. Add/Remove Snap-in screen gets displayed.

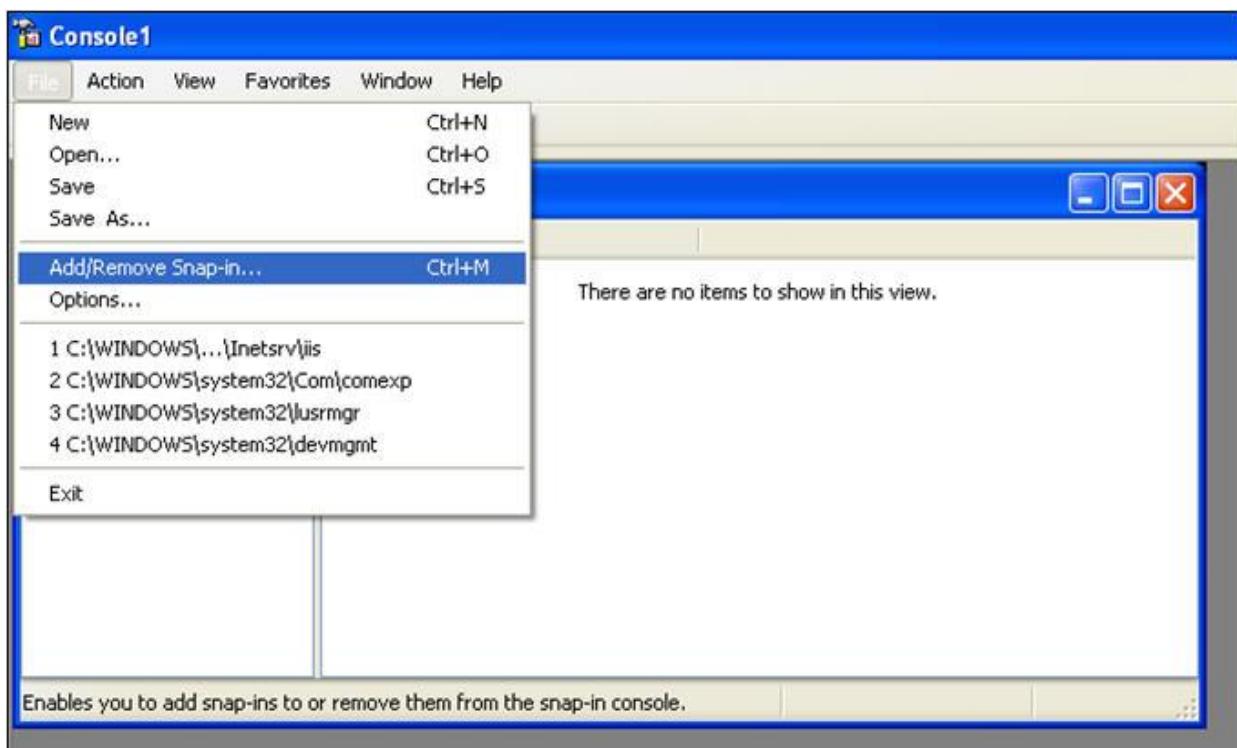


Figure 14-2 – Add/Remove Snap-in

- ☞ Click **Add** button. The Add Standalone Snap-in screen gets displayed.
- ☞ Select **Certificates**. Click **Add** button, the Certificates snap-in screen gets displayed.

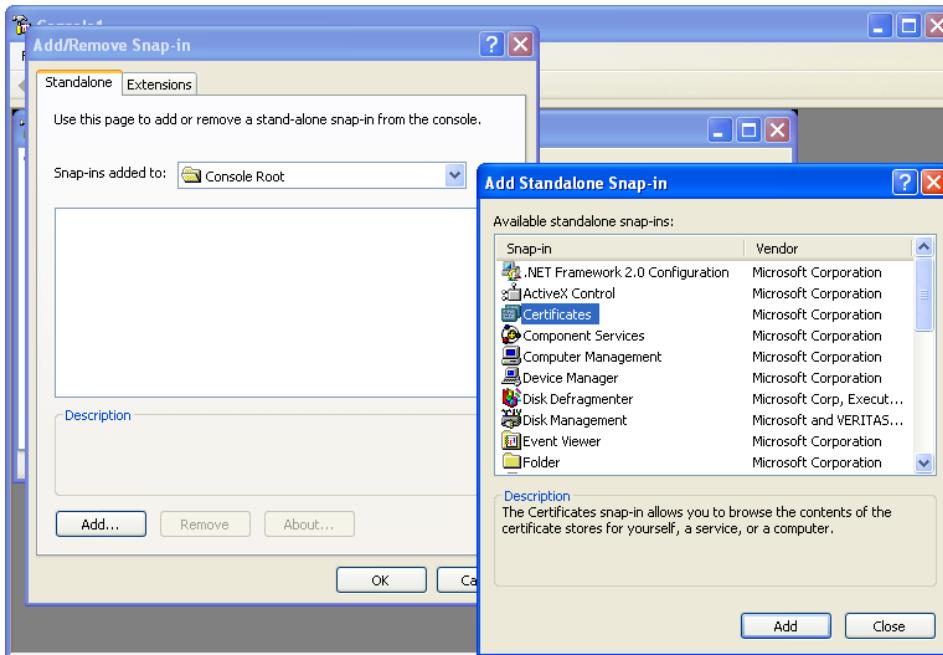


Figure 14-3 – Add Standalone Snap-in

- ☞ Select **Computer Account** radio button. Click **Next** button.

- ☞ Click **Finish** button. The Console Root screen gets displayed. Certificates snap gets added to the console root and lists all certificates to the local system.

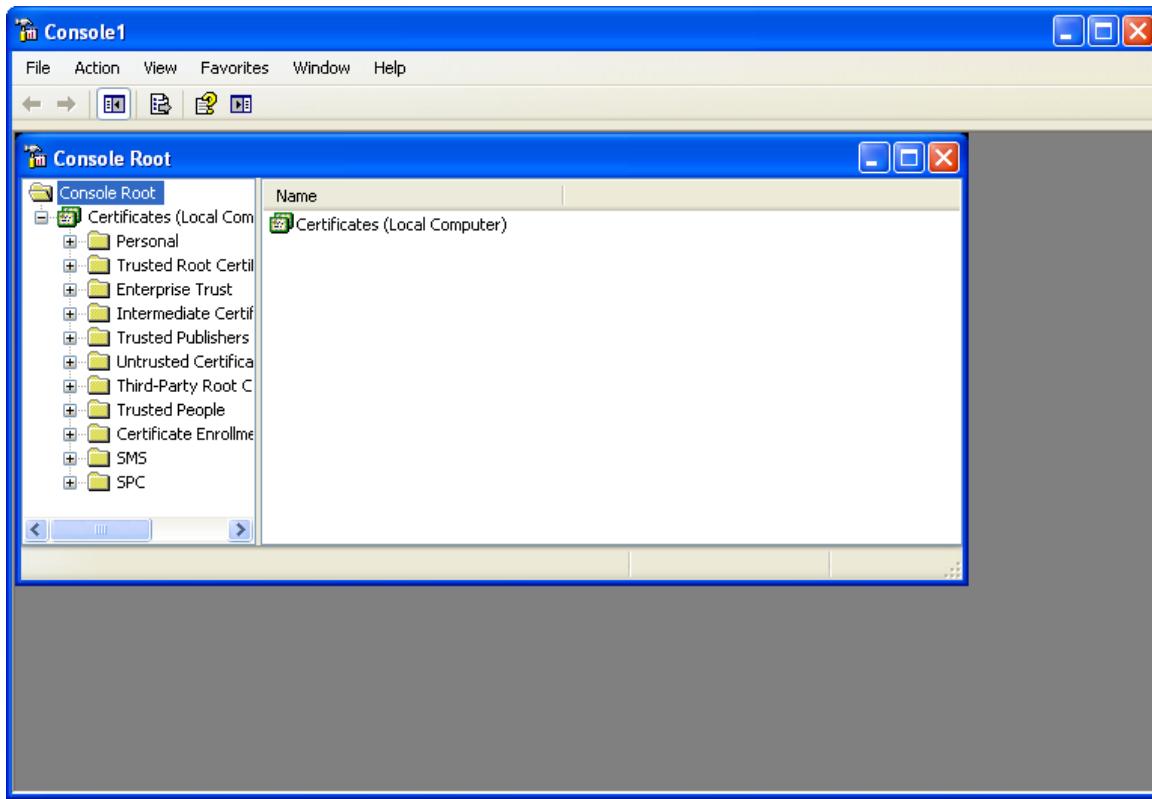


Figure 14-4 – Console Root

- ☞ Click Trusted Root Certification authorities and select Certificates.
☞ Right click Certificates>All task>Import.

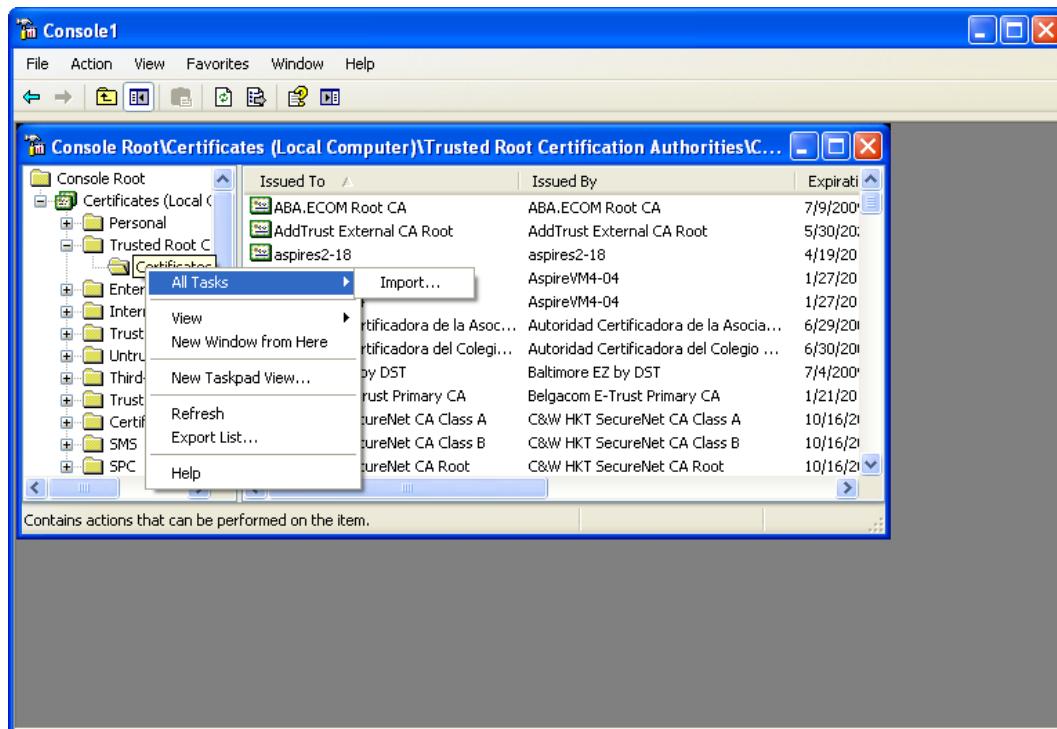


Figure 14-5 – Console Root

☞ Provide path of the saved certificate on the local system and click on **finish**.

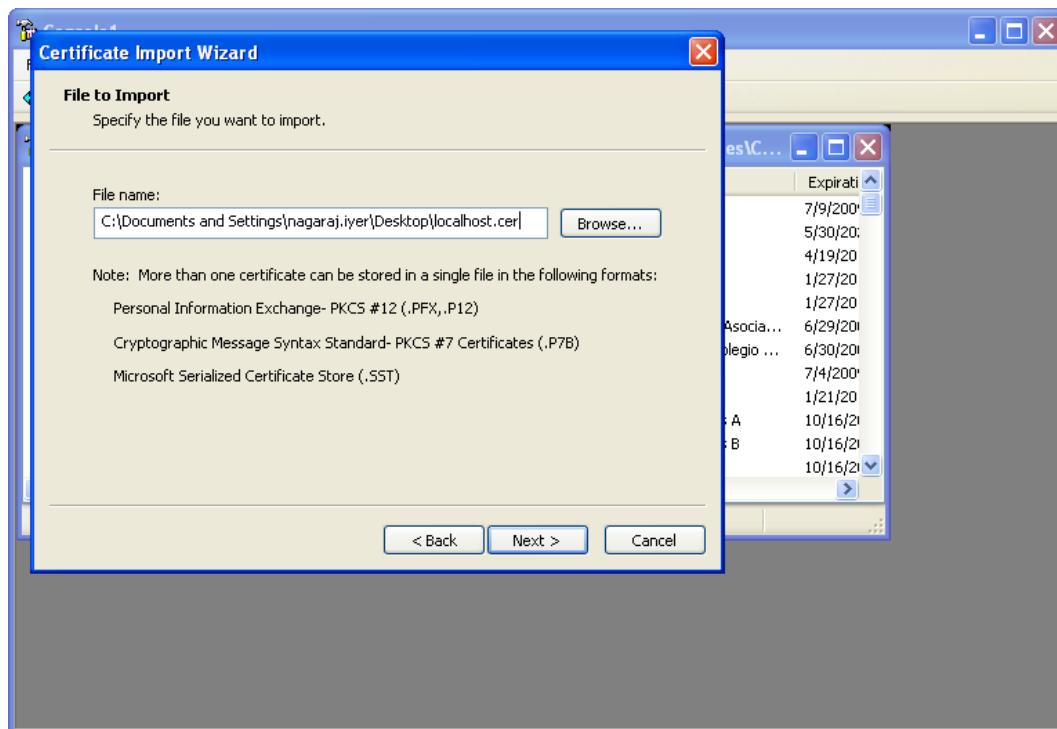


Figure 14-6 – Certificate Import Wizard

☞ The certificate will get added in trusted root certification authorities. This certificate is required for client authentication.

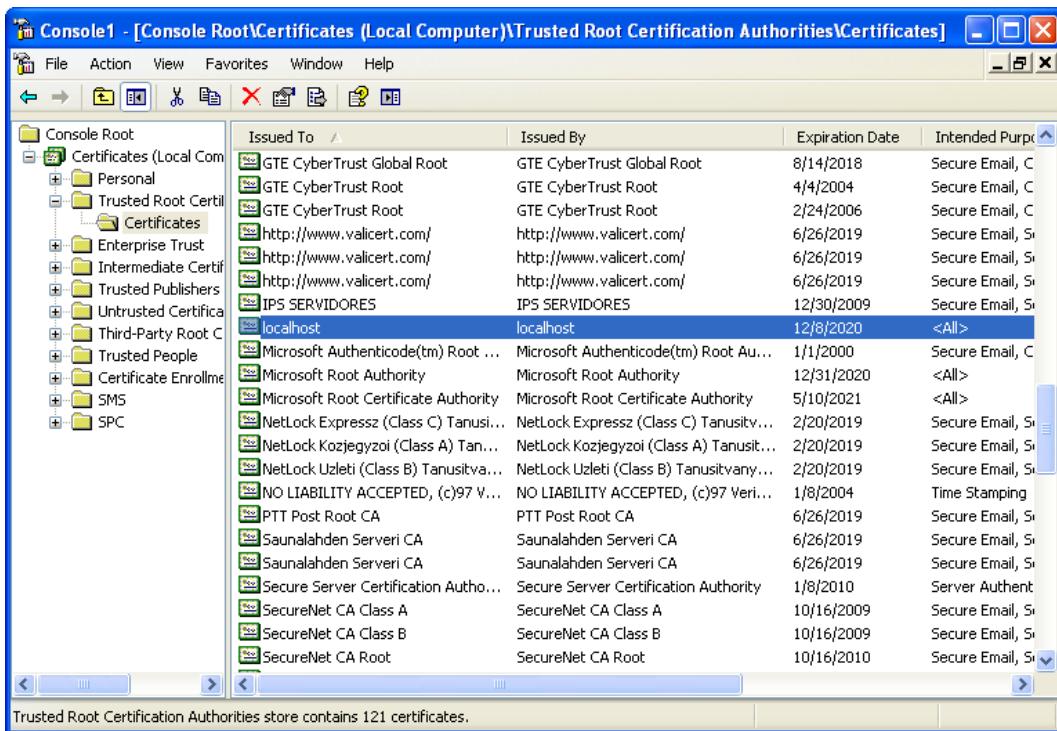


Figure 14-7 – Console

- ☞ TransportWithMessageCredential security mode is used in CelloSaaS WCF Service. You need to specify the certificate name in the web config 'serverCertificate' section.

```
<serviceCertificate findValue="localhost" storeLocation="LocalMachine" storeName="Root" x509FindType="FindBySubjectName"/>
```

- ☞ Create a virtual directory for the service, (say CelloSaaSWcfService).
- ☞ Provide a **server certificate** for the default web site hosted on the system (since the service uses transport security mode).
- ☞ Right click **Default web site > Properties > Directory Security > Server certificate**. You can use the same certificate that we used in our application.
- ☞ Right click on CelloSaaSWcfService > Properties > Directory security. Click Edit button in secure communication section.

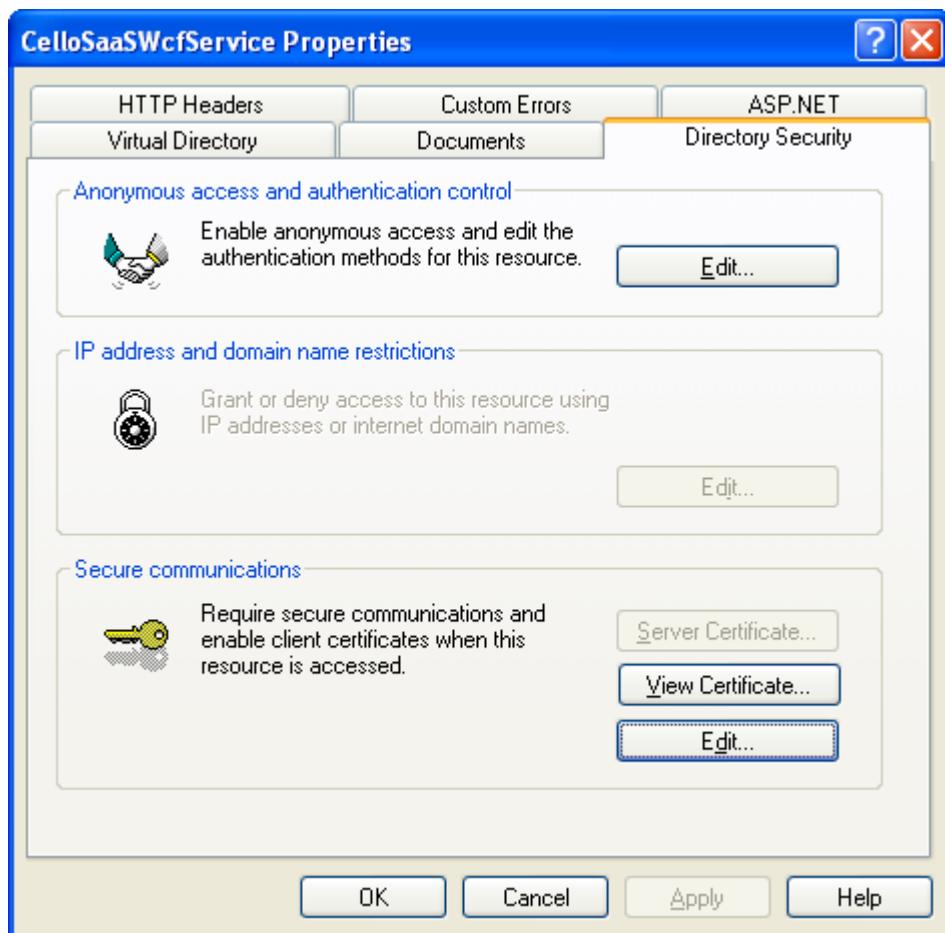
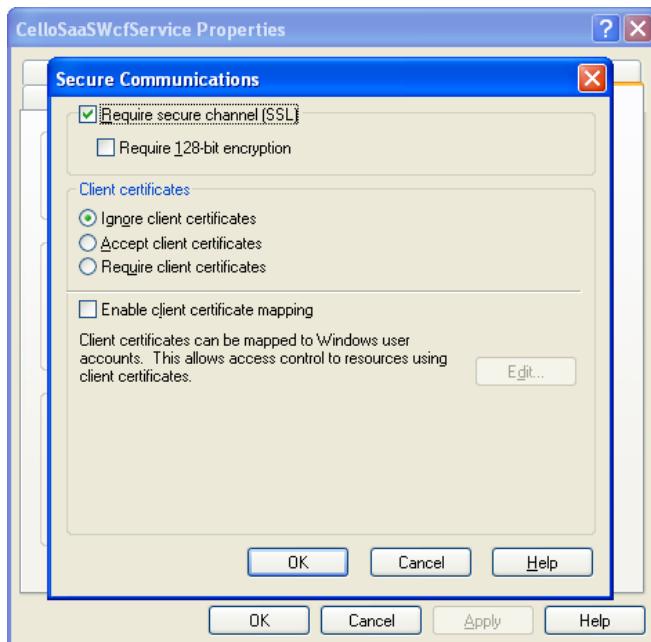


Figure 14-8 – Service Properties

- ☞ Click **"Require secure channel (SSL)"** checkbox in Secure Communications dialog box.
- ☞ Apply changes. Right click on the application and Click on **browse**.
- ☞ You will get an error saying "The page must be viewed on a secure channel". This means the configuration is working fine.
- ☞ Type the URL of your application in a browser with https:// and you will be able to see the service details.



The page must be viewed over a secure channel

The page you are trying to view requires the use of "https" in the address.

Please try the following:

- Try again by typing **https://** at the beginning of the address you are attempting to reach.

HTTP 403.4 - Forbidden: SSL required
Internet Information Services

Technical Information (for support personnel)

- Background:**
This error indicates that the page you are trying to access is secured with Secure Sockets Layer (SSL).
- More information:**
[Microsoft Support](#)

Figure 14-9 – Secure Communications

Creating service client

☞ Add service reference of the required service in your application.

CelloSaaS uses CustomAuthentication hence, you need to provide the credentials (your username and password) while creating an instance of service.

Sample

```
client.ClientCredentials.UserName.UserName = "admin@company.com";
client.ClientCredentials.UserName.Password = "password";
```



Since we are using certificates that are not signed by a valid certification authority, it will throw an error ("secure communication error") while trying to consume the service in development environment.

You can add the code snippet provided below to ignore this error.

```
System.Net.ServicePointManager.ServerCertificateValidationCallback += (se, cert, chain, sslerror)
=>
{
returntrue;
};
```

Sample code for fetching tenant details

```
System.Net.ServicePointManager.ServerCertificateValidationCallback += (se, cert, chain, sslerror)
=>
{
returntrue;
};
List<Tenant> tenantDetails;
WcfTenantService.TenantServiceClient client=new WcfTenant Service.
TenantServiceClient();
client.ClientCredentials.UserName.UserName = "admin@company.com";
client.ClientCredentials.UserName.Password = "company";
tenantDetails = client.GetAllTenantInfoDetails().ToList()
```

Wcf Urls

Access Control Management

<https://localhost/CelloWcfServiceApplication/AccessControlManagement/AccessControlService.svc>
<https://localhost/CelloWcfServiceApplication/AccessControlManagement/DataAccessService.svc>
<https://localhost/CelloWcfServiceApplication/AccessControlManagement/EntityPrivilegeService.svc>
<https://localhost/CelloWcfServiceApplication/AccessControlManagement/FeatureService.svc>
<https://localhost/CelloWcfServiceApplication/AccessControlManagement/PrivilegeService.svc>
<https://localhost/CelloWcfServiceApplication/AccessControlManagement/RoleService.svc>

Authentication Management

<https://localhost/CelloWcfServiceApplication/AuthenticationManagement/AuthenticationService.svc>

Business Rule Management

<https://localhost/CelloWcfServiceApplication/BusinessRuleManagement/RuleService.svc>

Configuration

<https://localhost/CelloWcfServiceApplication/Configuration/PickupListService.svc>

DataManagement

<https://localhost/CelloWcfServiceApplication/DataManagement/DataManagementService.svc>

License management

<https://localhost/CelloWcfServiceApplication/LicenseManagement/LicenseService.svc>
<https://localhost/CelloWcfServiceApplication/LicenseManagement/MeteringService.svc>

Master Data

<https://localhost/CelloWcfServiceApplication/MasterData/CountryService.svc>
<https://localhost/CelloWcfServiceApplication/MasterData/AddressService.svc>



Settings Management

<https://localhost/CelloWcfServiceApplication/SettingsManagement/RoleSettingsService.svc>
<https://localhost/CelloWcfServiceApplication/SettingsManagement/SettingsMetaDataService.svc>
<https://localhost/CelloWcfServiceApplication/SettingsManagement/TenantAuthenticateSettingService.svc>
<https://localhost/CelloWcfServiceApplication/SettingsManagement/TenantSettingsService.svc>
<https://localhost/CelloWcfServiceApplication/SettingsManagement/UserSettingsService.svc>
<https://localhost/CelloWcfServiceApplication/SettingsManagement/PackageSettingsService.svc>

Tenant Management

<https://localhost/CelloWcfServiceApplication/TenantManagement/TenantRelationService.svc>
<https://localhost/CelloWcfServiceApplication/TenantManagement/TenantService.svc>
<https://localhost/CelloWcfServiceApplication/TenantManagement/TenantTypesService.svc>

Tracking

<https://localhost/CelloWcfServiceApplication/Tracking/TrackingService.svc>

User Management

<https://localhost/CelloWcfServiceApplication/UserManagement/UserDetailsService.svc>

ViewMetaData

<https://localhost/CelloWcfServiceApplication/ViewManagement/ViewMetaDataService.svc>

14.1 Secret shared key for invoking services

This is a tenant wise setting of the secret key. The tenant can set a WCF shared key in the tenant setting page. This key will be used to authenticate CelloSaaS WCF service instead of sending password in user credential.

Since this is a secret key, the input is provided via the password field in the Tenant Settings view page.

The WCF Shared key is either hashed or encrypted and stored in the database. In the case of a hashed key, the value cannot be decrypted.

In CelloSaaS wcf application, we have to change customUserNamePasswordValidatorType to "CelloSaaS.Services.WCF.SecretKeyClientValidator" in userNameAuthentication

```
<serviceBehaviors>
<behavior name="ServiceBehavior">
<serviceMetadata httpsGetEnabled="true" httpGetEnabled="false" />
<serviceCredentials>
<userNameAuthentication userNamePasswordValidationMode="Custom" customUserNamePasswordValidatorType="CelloSaaS.Services.WCF.SecretKeyClientValidator, CelloSaaS.Services" />
<serviceCertificate findValue="TestWCFHost" storeLocation="LocalMachine" storeName="Root" x509FindType="FindBySubjectName" />
<clientCertificate>
<authentication certificateValidationMode="None" />
</clientCertificate>
<issuedTokenAuthentication allowUntrustedRsaIssuers="true" />
</issuedTokenAuthentication>
</serviceCredentials>
<serviceDebug includeExceptionDetailInFaults="true" />
</behavior>
</serviceBehaviors>
```

15 CELLOSAAS WORKFLOW

15.1 Workflow

Workflow

Workflow is a term used to describe the tasks, procedural steps, organizations or people involved, required input and output information, and tools needed for each step in a business process. A workflow approach to analyzing and managing a business process can be combined with an object-oriented programming approach, which tends to focus on documents and data. In general, workflow management focuses on processes rather than documents. For example, an insurance company could use a CelloSaaS workflow Engine to ensure that a claim was handled consistently from initial call to final settlement. The CelloSaaS workflow Engine would ensure that each person handling the claim used the correct online form and successfully completed their step before allowing the process to proceed to the next person and procedural step.

Workflows

Workflows can be best understood in either of the following definitions

Def 1. A workflow is a *model*, which means it is a machine-readable description of business behavior that is not code.

Def 2. A workflow is a system that manages and defines a series of tasks within an organization to produce a final outcome or outcomes [1].

Workflow management systems allow the user to define different workflows for different types of jobs or processes [2].

Use Case 1:

For example, in a manufacturing setting, a design document might be automatically routed from designer to a technical director to the production engineer.

At each stage in the workflow, one individual or group is responsible for a specific task [3]. Once the task is complete, the workflow ensures that the individuals responsible for the next task are notified and receive the data they need to execute their stage of the process. Workflow management systems also automate redundant tasks and ensure that uncompleted tasks are followed up.

Use Case 2:

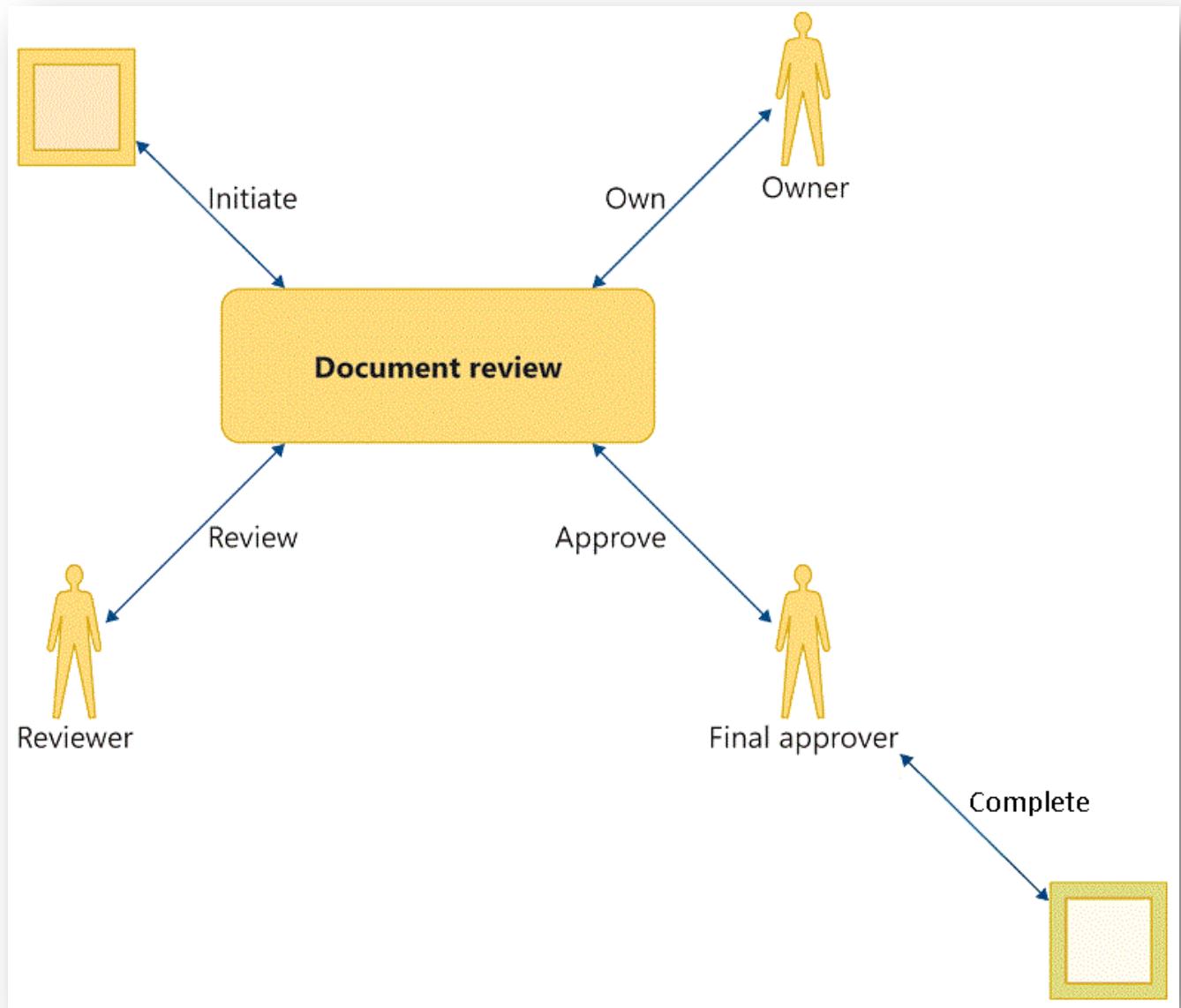
A workflow model describes an organization of *work units* [4].

For instance, suppose that a document review process specifies that Joe writes the document and then Fred reviews it. Here, the work units are first writing and second reviewing the document, and the organization is that one task must follow the other. Code that makes successive calls to two subroutines is a valid example of the concept. The interest lies rather in the forms that this organization takes.

A *document review* process takes as an input parameter [5] a set of [reviewer, role] pairs that describe which humans are involved in the workflow in which roles [6]. Possible values for the role

are required, optional, final approver, and owner. The review process then proceeds until all reviewers have performed their assigned roles and notify the owner of the outcome.

Here, the work items are the document reviews organized by the review process [7]. There are three interesting characteristics to call out, namely, multiple points of interaction [8], human [9] and automated activity [10], and the need to handle dynamic change.



Sample Document Review Workflow

Pointers

- [1] Each activity has a set of outcomes. These activities and their outcomes are mapped in the Activities Table in CelloSaaS.
- [2] Cello allows Tenant based custom workflow design for a single workflow and also maintain different workflows for different tasks. Also workflow can be grouped by custom categories.
- [3] This requirement is captured via the task actors or dynamic task actors. There are two options, namely usernames based user assignment or role based user assignment.

[4] These are the tasks which are associated with the activities like manual / automated task with their corresponding activities.

[5] Each task can take inputs and is defined by the developer and is made available via Inputs to activities in a task.

[6] Cello identifies the users by either usernames or the users that belong to a role, identified by role names.

[7] These process can be globally defined and tenant-wise customizable. If not customized, the workflow engine defaults to the global process [workflow].

[8] There are routers that decide on what step to proceed next based on the outcomes of the previous conditions. Cello has different types of Routers to accomplish this feature.

[9] This is achieved via the Manual Task. This will require a manual intervention to process the activity and to move it to next step.

[10] This is achieved via the Automatic / Automated Task. No manual intervention is required. This can be an auto approval based on the outcome of the previous task.

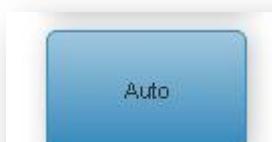
CelloSaaS Workflow

- 1 CelloSaaS provides design time and runtime support for workflow management.
- 2 A workflow can be invoked based on the following
 - a Occurrence of an event
 - b Invocation in the code as part of a business process
- 3 Workflows should be uniquely identified
- 4 Workflows will have an input on which the workflow acts

Steps in a Workflow

- 1 Workflow is an orchestration of steps crafted to portray the business process
- 2 Every workflow begins with a **start** step and ends with a **stop** step
- 3 A step can either be manual or automatic
- 4 A step must be mapped to an activity and can further have provisions to associate task or code conditions for Skip, Start, Expiry and Complete conditions. All the conditions can either be
 - o Task outcome
 - o Execution based conditions
 - o Code condition
 - o Business Rule conditions.
- 5 Every step can have one or more outcomes returned by the corresponding activities.
- 6 The step status can
 - o Routed : Step is routed and is awaiting processing by the next step
 - o Skipped: This step is skipped from execution based on the step's associated condition
 - o Started: This step has started processing
 - o Executed: This step has completed execution
 - o Errorred: Some exception has occurred impeding the execution of this step
 - o Completed: This step has run to completion
 - o Expired: This step failed to execute / complete within the specified expiry condition and is considered to be expired. This step can be re-started or revived manually.
- 7 A manual step can also contain definition for who all should act on that step. This can either be static roles or dynamic users.

Tools

Tool	Description
	<p>Start Step: Every workflow begins with a Start Step and this step is mandatory</p>
	<p>Manual Task represents the task that requires user intervention. This task also obtains the users that will be required to act on the task.</p>
	<p>Auto-Task represents the task that can execute based on the previous task outcome or conditions without requiring user intervention.</p>
	<p>Router is used to route the flow to tasks based on some conditions.</p>
	<p>If-Else-Router is a specialized form of Router that has if-else branching construct for the conditions. If the condition is true then task which is connected with If node will be executed and vice-versa..</p>
	<p>End Step: Every workflow ends with this End Step. This step is mandatory as in the case with the Start Task.</p>

Tasks

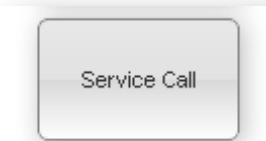
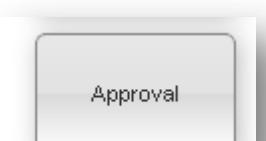
CelloSaaS supports the following types of tasks in a workflow.

- 1 Manual Task
 - a This refers to the task that has to require a manual / user intervention to continue processing.
- 2 Auto Task
 - a This does not require a manual / user intervention to process.

Activity

A task contains a collection of activities defining the order in which the user should interact with. The Activities toolbar lists all the activities that are registered with Workflow. A task will refer to an activity which could be any kind of action. For Example: Sending email, Update in the database, Processing document or image etc.

- All possible activities need to be thought through and developed and registered
- An activity can be used in any number of steps within a workflow
- The stored procedures that can be utilized for the activities should also be registered

Activity Tools *	Description
	<p>This Activity uses the CelloSaaS notification manager to send an email Notification based on the configured properties in this activity</p>
	<p>This activity will make a service call based on the registered web service URL with the inputs and format the outputs for further use by workflow engine.</p>
	<p>Performs the approval or reject action on the input based on the task conditions</p>

* - Custom activities that may be developed and registered with CelloSaaS will also be listed here.

Note on Activity Inputs

The inputs that may be passed to an activity can be either objects or XML data. In case of an In-Proc mode, the data can be available in object format and in the case with Web Services, the inputs can be in XML format and CelloSaaS supports both these modes.

Default Activities

The following are the default activities shipped out of the box in CelloSaaS,

1. Email Notification Activity

- a. This activity can be used to trigger an email notification.
- b. The following are the properties
 - i. To, CC : The inputs will typically be a comma delimited string or string array or a list of email ids. Sample: user1@example.com, user2@example.com
 - ii. ToXPath, CCXPath: Typical inputs will be a XPath query to obtain the To, CC email ids from the Workflow input in case of the input being a WorkflowXmlInput. Sample: //userDetails /MembershipDetails/ EmailId
 - iii. To Property, CC Property: In this property, we can provide the property of the application entity that can provide the values for the To / CC property.
Sample: userDetails.MembershipDetails.EmailId

Task Definition

Task Details	Conditions	Properties	Outcomes & Status
Property	Value		
Notification Name	Approval Mail		
Subject	Document is approved after Review		
To	userone@example.com		
To xPath	//userDetails /MembershipDetails/ EmailId		
To Property	userDetails.MembershipDetails.EmailId		
Cc			
Cc xPath			
Cc Property			

Save Cancel

2. Approval Activity

- a. This activity can be used to either approve or reject a task.
- b. There are no properties involved in this activity as this activity can either approve or reject

3. Service Call Activity

- a. A web service can be invoked by providing the fully qualified service path in the Service Name property.
- b. The Service Input Transform is used to input the XSLT that can be used to transform the workflow input to service call input.
- c. The Workflow input transform is used to input the XSLT that can be used to transform the service call output to workflow input.

Task Definition

Task Details	Conditions	Properties	Outcomes & Status								
<table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Service Name</td> <td>get_acct_message</td> </tr> <tr> <td>Service Input Transform</td> <td> <pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" encoding="utf-8" indent="yes"/> <xsl:template match="/"> <xsl:apply-templates select="//SubjectXmlValue"/> </xsl:template> <xsl:template match="//SubjectXmlValue/PROV_MESSAGE"> <root> <acct_no> <xsl:value-of select="substring-after(substring-before(.,'UserID'))'Account Number: ') /> </acct_no> <message_id> <xsl:value-of select="substring-after(substring-before(.,'</pre> </td> </tr> <tr> <td>Workflow Input Transform</td> <td> <pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" encoding="utf-8" indent="no"/> <xsl:template match="/"> <xsl:copy-of select="*"/> </xsl:template> </xsl:stylesheet></pre> </td> </tr> </tbody> </table>				Property	Value	Service Name	get_acct_message	Service Input Transform	<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" encoding="utf-8" indent="yes"/> <xsl:template match="/"> <xsl:apply-templates select="//SubjectXmlValue"/> </xsl:template> <xsl:template match="//SubjectXmlValue/PROV_MESSAGE"> <root> <acct_no> <xsl:value-of select="substring-after(substring-before(.,'UserID'))'Account Number: ') /> </acct_no> <message_id> <xsl:value-of select="substring-after(substring-before(.,'</pre>	Workflow Input Transform	<pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" encoding="utf-8" indent="no"/> <xsl:template match="/"> <xsl:copy-of select="*"/> </xsl:template> </xsl:stylesheet></pre>
Property	Value										
Service Name	get_acct_message										
Service Input Transform	<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" encoding="utf-8" indent="yes"/> <xsl:template match="/"> <xsl:apply-templates select="//SubjectXmlValue"/> </xsl:template> <xsl:template match="//SubjectXmlValue/PROV_MESSAGE"> <root> <acct_no> <xsl:value-of select="substring-after(substring-before(.,'UserID'))'Account Number: ') /> </acct_no> <message_id> <xsl:value-of select="substring-after(substring-before(.,'</pre>										
Workflow Input Transform	<pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" encoding="utf-8" indent="no"/> <xsl:template match="/"> <xsl:copy-of select="*"/> </xsl:template> </xsl:stylesheet></pre>										
<input type="button" value="Save"/> <input type="button" value="Cancel"/>											

Outcomes

The following are the possible outcomes for the default activities from CelloSaaS.

Activity Name	Possible Outcomes
Email Notification	Complete
Service Call	Complete
Approval	Approve, Reject

Step Router

- The step router is responsible for routing the flow between the steps in workflow

- This routing is based on task conditions which are again based on the previous step outcomes and execution status
- Step router can have multiple inputs and output flows.
- The router can have decisions which is a combination of outcomes from two tasks
- Parallel routing is possible
- Re-routing to a finished task is also possible

A sample router with the corresponding code expression condition is given below

Router Definition

Router Details Conditions

Router Name: Training Approved

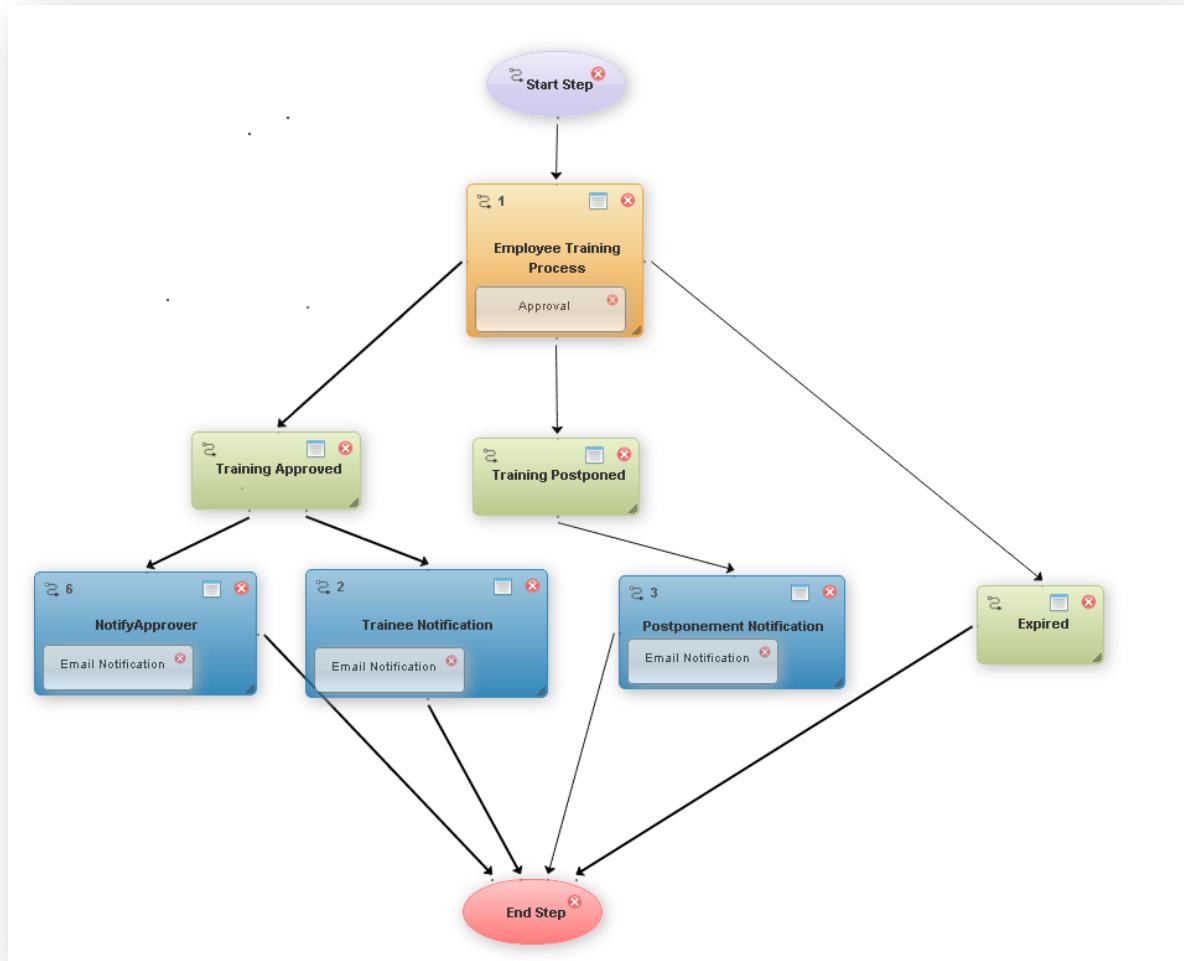
Router Definition

Router Details Conditions

Task Condition Expression **Add Condition**

TaskExpressionCondition : O:EmpTrainProcess.Approve

The following sample training workflow shows the parallel routing in a Router



Workflow Definition

- A product administrator will be able to define the default steps of a workflow
- The tenant administrator will be able to define a workflow for them with the steps registered
- The workflow can be versioned for each tenant
- The workflow version can be published and when reopened for editing it gets to the draft mode

Workflow Instance, Step Instance and Step routing

- When a workflow is invoked an instance of the workflow is created with a unique identification
- Each workflow status is tracked through various statuses that it may pass through as time progresses.
- Once the workflow instance starts it begins with the start step
- Once the start step is executed, it routes it to the router
- The router then evaluates the next set of steps that need to be routed to
- Once the steps are routed, the task instance gets to the status “Routed” and the workflow instance checks if the step can be skipped
 - If it can be skipped, the task instance moves to a status called Skipped and the corresponding routers linked to the task gets evaluated and then routed to.
 - If skip condition fails then the start task for this step will be called.
- If the step can be started, the task instance moves to a status called “started” and calls the execute action which internally executes the activity mapped to it.

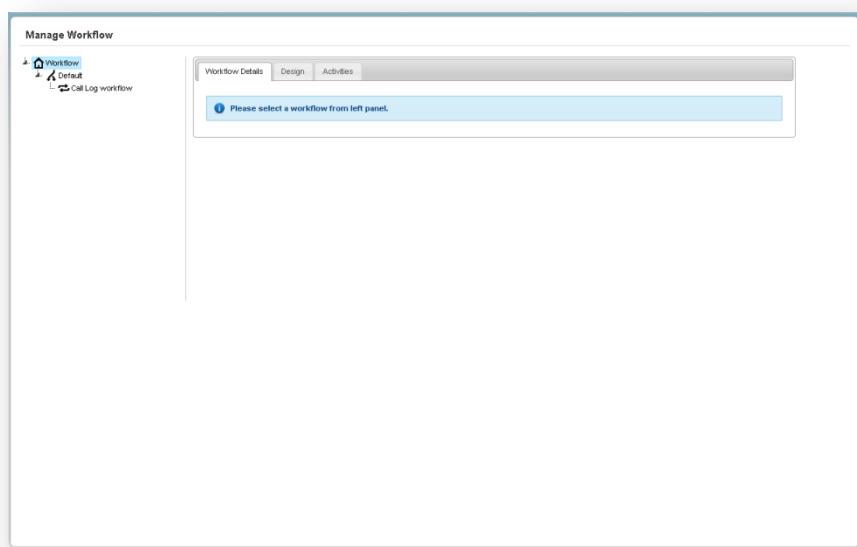
- The execute activity returns back the outcomes and the status is updated to “executed”
- Once the task is executed, the step is checked if it can be completed. If so, it completes the task and marks the task instance status as “Completed”.
- If any exception occurs while executing the task then the task status is set as “Errored”.
- When the end step is completed then the workflow instance gets completed and its status is marked as “Completed”.
- If task complete condition fails then expired conditions will be checked. If expire condition is satisfied then the task status moves to “Expired”.
- Whenever there is any change in the status or outcome of the task status then the corresponding routers linked to the task gets evaluated and then routed to.

Managing Workflows

Navigate to the Workflow Index page; there are two panels in this page,

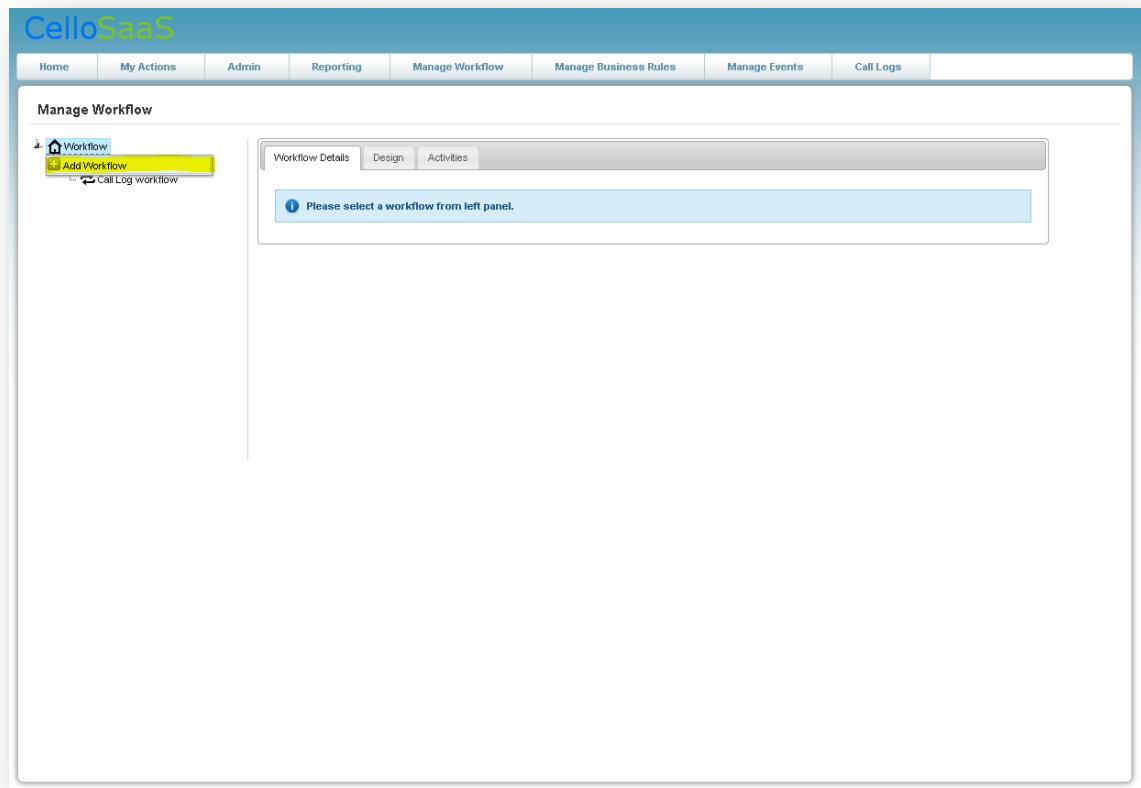
Left Panel: This panel lists all the available workflows

Right Panel: Lists the details about the selected workflow



Add Workflow

In order to add a new workflow, in the right panel, click on the workflow home icon and then right click on it to get the context menu. The context menu lists the possible actions, in this case, “Add Workflow”. Click on the “Add Workflow” context menu item to add a new workflow.



Workflows created from the above given UI are created on behalf of the logged in tenant only. The following screenshot illustrates the various inputs that are required to create a new workflow.

Input	Description
Name	Workflow name [Unique per tenant]
Description	Short description about the workflow
Workflow Input finder Type	The Input finder class that will get the workflow input when the workflow is to resume with the data from the database.
Category	Logical grouping of the workflow
XML Transform	The XSLT that is required to transform the XML input to the workflow input
Is Global	To indicate that this workflow is visible across all the tenants.

Adding a new workflow

Manage Workflow

Workflow Name	<input type="text"/>
Description	<input type="text"/>
Workflow Input Finder Type	<input type="text"/>
Category	<input type="text"/> Notification
XML Transform	1
Is Global	<input checked="" type="checkbox"/>

Save **Cancel**

Editing a workflow

Workflow Details

Workflow Name	<input type="text"/> Student Management Workflow
Description	<input type="text"/> Student Management Workflow
Workflow Input Finder Type	<input type="text"/> ezCLM,ezCLM,LMSWorkflowFind
Category	<input type="text"/> Employee
XML Transform	<pre> 1 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> 2 <xsl:output method="html" encoding="UTF-8" indent="no"/> 3 4 <xsl:template match="/"> 5 <table class="celloTable"> 6 <tbody> 7 <tr> 8 <td style="width:250px;"> 9 <label>Student Name</label> 10 <td> 11 <xsl:value-of select="/StudentDetails/Name"/> 12 </td> 13 </td> 14 </tr> 15 <tr> 16 <td> 17 <label>Student Email</label> 18 <td> 19 <xsl:value-of select="/StudentDetails/EmailId"/> 20 </td> 21 </tr> 22 <tr> 23 <td> 24 <label>No Of Days Leave</label> 25 <td> 26 </td> 27 </tr> </pre>

Save

Managing Workflow Designs

In this page, the various orchestrated workflows are listed,

1. Default Workflow Design

- a. This is the workflow design that is default and the tenant administrators can override this default design and create their own to suit their requirements. If the overridden one is not published, the default workflow is taken for execution.
2. Published Workflow Design
 - a. This is the list of designs that have been published by the current tenant admin along with the version numbers.
 3. Un-Published workflow design
 - a. This contains the designs that are yet to be published.

Workflow Details Design Activities

Manage 'Employee Training Workflow' **Create New** Filter Id: ---Select---

Workflow Name: Employee Training Workflow
Description: Employee Training Workflow
Active Version: 14

Default Workflow Design

Version	Publish Status	Created On	Updated On	Edit	Copy
1	Published	1/29/2013 9:46:28 AM	1/29/2013 11:32:36 AM		

Published Workflow Designs

Filter Id	Version	Created On	Updated On	Status	View	Copy
---	14	2/1/2013 10:59:16 AM	2/1/2013 12:01:50 PM	Active		
---	13	2/1/2013 10:50:33 AM	2/1/2013 10:51:25 AM	---		
---	12	2/1/2013 10:42:56 AM	2/1/2013 10:43:35 AM	---		
---	11	1/31/2013 4:17:23 PM	2/1/2013 10:38:39 AM	---		
---	10	1/30/2013 3:31:25 PM	1/30/2013 3:31:27 PM	---		
---	9	1/30/2013 2:34:46 PM	1/30/2013 2:35:07 PM	---		
---	8	1/30/2013 2:31:51 PM	1/30/2013 2:32:31 PM	---		
---	7	1/30/2013 12:10:20 PM	1/30/2013 2:25:00 PM	---		
---	6	1/29/2013 4:52:18 PM	1/30/2013 12:06:07 PM	---		
---	5	1/29/2013 3:37:33 PM	1/29/2013 3:38:27 PM	---		
---	4	1/29/2013 3:32:24 PM	1/29/2013 3:34:04 PM	---		
---	3	1/29/2013 1:49:48 PM	1/29/2013 1:55:38 PM	---		
---	2	1/29/2013 11:32:41 AM	1/29/2013 12:15:46 PM	---		

Un-Published Workflow Design

Filter Id	Version	Created On	Updated On	Action	Edit	Delete
---	15	2/1/2013 3:37:42 PM	--	Publish		

Activities

The following are the activity details for the default activities that are supplied with CelloSaaS out of the box

Workflow Details Design Activities

Email Notification

Description: Sends Notification
Type Name: CelloSaaS.WorkFlow.Activities.NotificationActivityDefinition
Assembly Name: CelloSaaS.WorkFlow
Outcomes: Complete
Properties: Notification Name , Subject , To , To xPath , To Property , Cc , Cc xPath , Cc Property

Service Call

Description: Calls a service through web service
Type Name: CelloSaaS.WorkFlow.Activities.ServiceCallActivityDefinition
Assembly Name: CelloSaaS.WorkFlow
Outcomes: Complete
Properties: Service Name , Service Input Transform , Workflow Input Transform

Approval

Description: Approval Activity
Type Name: CelloSaaS.WorkFlow.Activities.ApprovalActivityDefinition
Assembly Name: CelloSaaS.WorkFlow
Outcomes: Approve,Reject
Properties: -

Manual and Auto Task Definitions

The difference between a manual and an automatic task definition is that the manual task definition involves the actor detail capture to perform the manual task which is not the case with the automatic task.

Manual Task Definition

Task Definition

Task Details Actor Details Conditions Properties Outcomes & Status

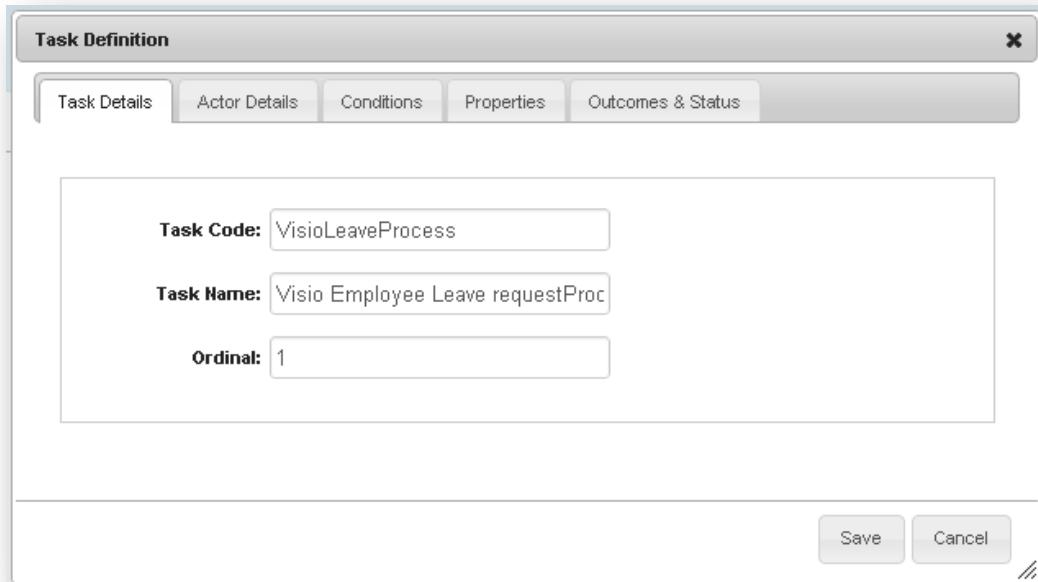
Auto Task Definition

Task Definition

Task Details Conditions Properties Outcomes & Status

Task Definition Properties

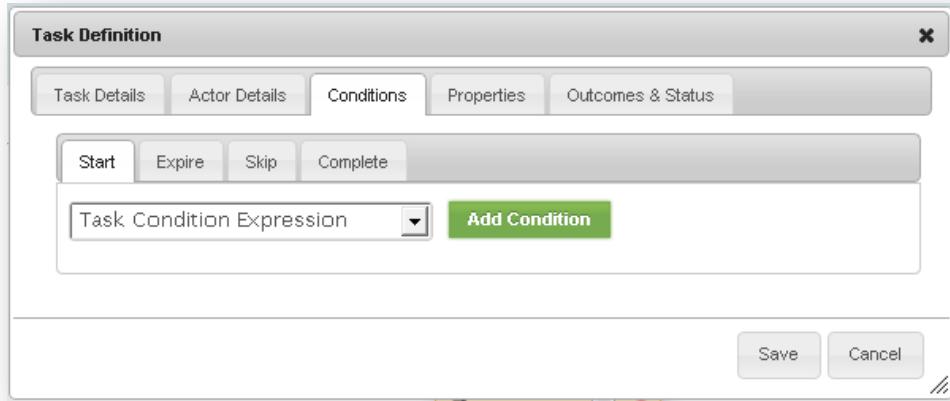
Task	Description
Task Code	This is the unique task code in the workflow.
Task Name	This is the Name of the Task
Ordinal	This is the ordinal value of the Task. Default ordinal value will set while drag and drop. The user also can edit and change the ordinal values as like.



Task Definition Properties

Task	Description
Conditions	<p>It has the conditions to execute on different process of the workflow execution like Start, Expire, Skip, and Complete. Each has the conditions for own</p> <ul style="list-style-type: none"> ○ The Start conditions will check before the task started. If the condition is true the task will get start. ○ The Expire conditions will check before the task expired ○ The Skip condition will check the condition while skips the task. ○ The Complete condition will check the condition while the task is complete.
Conditions Type	<p>Task Condition:</p> <ul style="list-style-type: none"> ○ It has the condition while the task is process. It has two types as outcomes and executions status. ○ The Outcome based condition will be like "<i>O:[TaskName].[Outcome]</i>" <p><i>O</i> → outcome <i>[TaskName]</i> → Task name <i>[Outcome]</i> → Task possible outcomes from Execute and Complete methods.</p> <ul style="list-style-type: none"> ○ The Execution Status based condition will be like "<i>E:[TaskName].[ExecutionStatus]</i>" . <p><i>E</i> → Execution Status <i>[TaskName]</i> → Task name <i>[ExecutionStatus]</i> → Execution status. Possible execution status → Started, Skipped, Routed, Errorred, Executed, Expired, Completed.</p>

	<p>Code Condition:</p> <ul style="list-style-type: none"> ○ The user can have their own conditions like "Systems.DateTime.Now > Convert.ToDateTime(dateTime)" which is plain C# code. ○ Entity also can be used in this condition. <p>Stored Procedure Condition:</p> <ul style="list-style-type: none"> ○ We are not supporting this feature in this version.
--	--

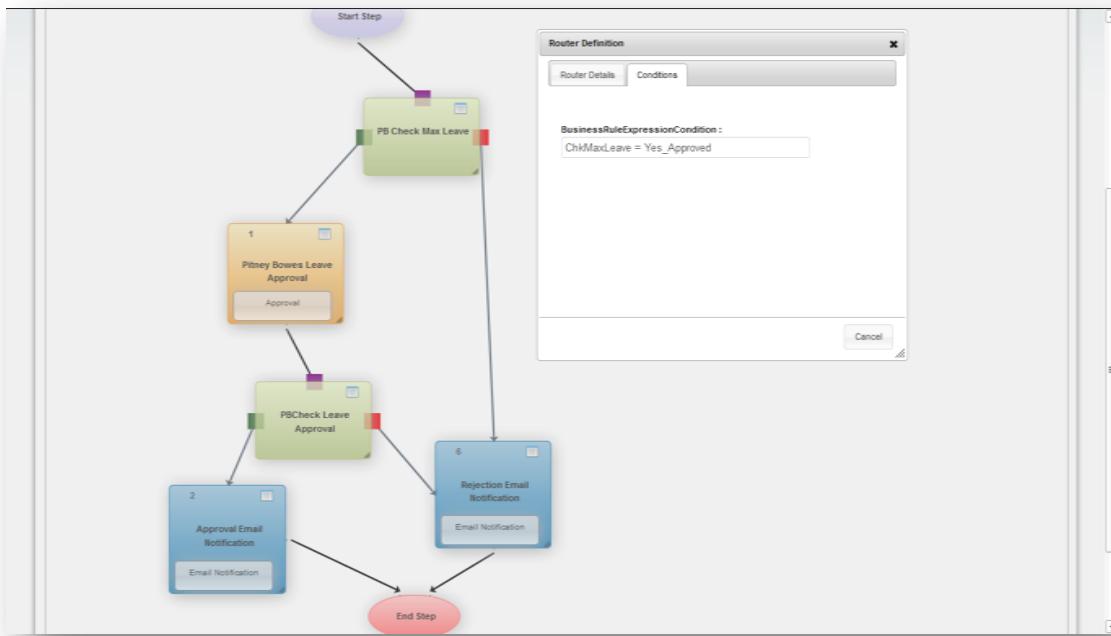


Sample Business Rule Condition

1. Business Rule Creation

Rule Details	Rule Activities		
RuleSet Code	ChkMaxLeave	RuleSet Name	ChkMaxLeave
Description	Business Rule to approve/reject based on No of leaves	Category	Default
Screen Type	ConditionAction	Is Global	True
<input type="button" value="Test"/> <input type="button" value="Configure"/>			

2. Mapping the Rule to the Workflow router condition



Sample Code Expression Condition

Task Definition

Task Details Actor Details Conditions Properties Outcomes & Status

Start Expire Skip Complete

Code Condition Expression ✖

CodeExpressionCondition :
System.DateTime.Now >= Convert.ToDateTime("03/04/2013 12:15:00")

Save Cancel

Task Definition Properties

Task	Description
Outcomes and Status	The Outcomes and Status contains the possible outcomes list and possible execution status list. This outcomes and Executions status list will define while the activity creation

Task Definition

Step Execution Status:

Step Execution Status can be used in Task Condition expression.
 Example: **E:Task Code.Completed**

Possible step execution status are:

- Started
- Skipped
- Routed
- Errored
- Executed
- Expired
- Completed

Activity Outcome Details:

Activity outcomes can also be used in Task Condition expression.
 Example: **O:Task Code.Approved**

Activity Name	Possible Outcomes
Email Notification	Complete
Service Call	Complete
Approval	Approve, Reject

Save **Cancel**

Task Definition Properties

Task	Description
<i>Actor Details (only for Manual Task)</i>	It has <i>Roles</i> , <i>Role Names</i> , <i>User Names</i> and <i>Url</i> attributes. Roles textbox we will give the role identifiers with comma separated. Role Names textbox we will give the role names. User Names textbox we will give the user names of the tenant. The <i>Url</i> is used to execute the Task with use of given <i>Url</i> value.

Task Definition X

Task Details Actor Details Conditions Properties Outcomes & Status

Roles:
(Enter role ids separated by ',')

Role Names:
(Enter role names separated by ',')

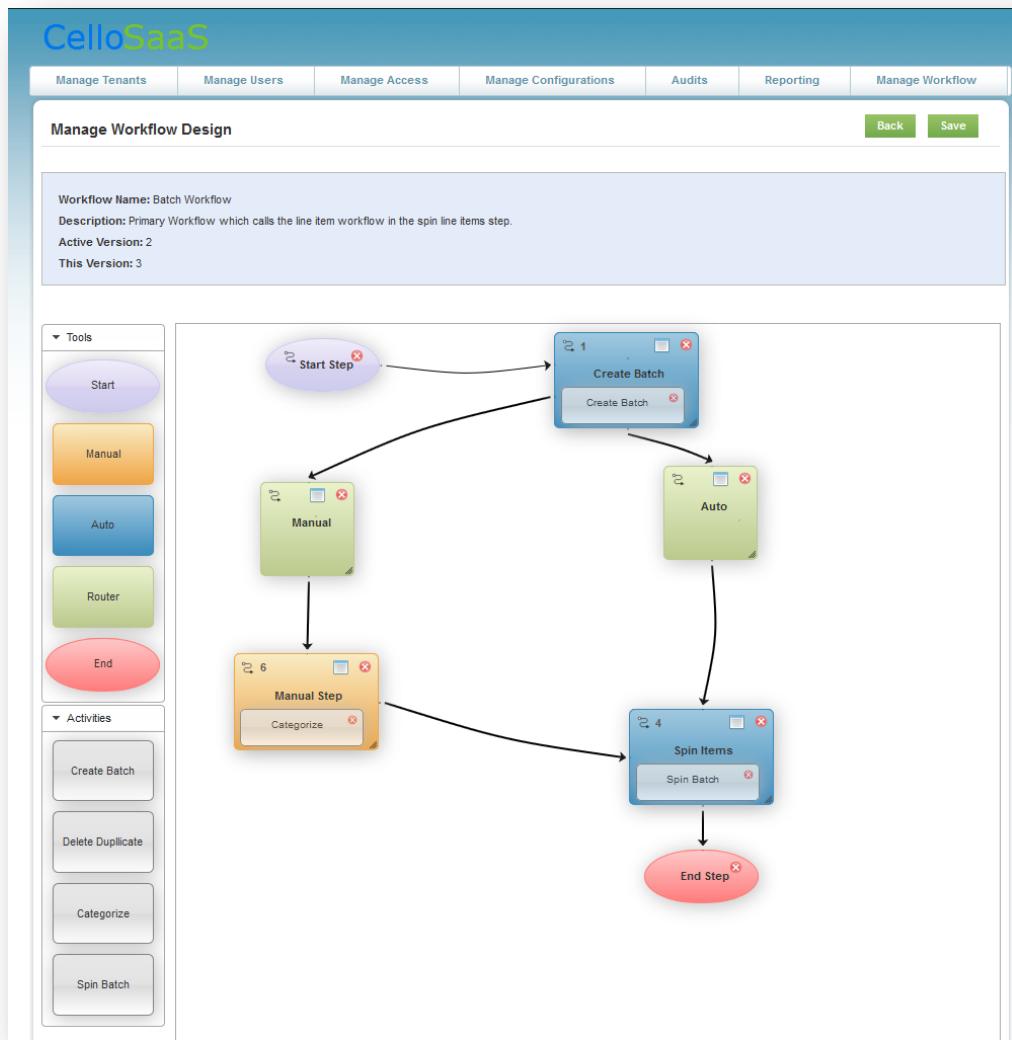
User Names:
(Enter user names separated by ',')

Url:

Save Cancel / / / / /

Workflow Design

Workflow design contains the Tools and Activities in the sidebar. Using the tools and the Activities the tenant/user can design their own workflow and save it.



Activity Creation

How to create activity

Create activity class that implements the CelloSaaS.WorkFlow.Model.IActivity interface.

```

public class SendingEmailActivity : IActivity
{
    public List<object> Inputs { get; set; }

    public ITaskActivityDefinition ActivityDefinition { get; set; }

    public bool CanStart()
    {
        return true;
    }

    public List<string> Execute()
    {
        // Send Email Logic.
        return new List<string>() { "EmailSend" };
    }
}

```

```

public bool CanComplete()
{
    return true;
}
public List<string> Complete()
{
    // Write logic after send the mail
    return new List<string>() { "Completed" };
}
public bool HasExpired()
{
    return false;
}
}

```

Members	Description
Inputs	This is used to assign the input values to the Activity from the application/service.
ActivityDefinition	This is used to gets or sets the Activity definition details.
CanStart()	This method is used to check whether can start the task or not. Write can start logic in this method. If this method returns “True” then only the task will be execute
Execute()	This method is used to execute your business logic for this task. After executed the business logic return the outcomes (outputs).
CanComplete()	This method is used to check whether can complete the task or not. Write can complete logic in this method. If this method returns “True” then only the task will be completed.
Complete()	This method is used to write the task complete logic. Retune the outcomes (output) after the logic executed

Creating an Activity Definition

- CelloSaaS supports two type of Activity Definition.
 - DLL Task Activity Definition
 - Sp Task Activity Definition
- Create class that inherits from CelloSaaS.WorkFlow.Model.DIITaskActivityDefinition, if the activity is DLL based. If the activity is SP based then inherit from CelloSaaS.WorkFlow.Model.SPTaskActivityDefinition.
- Create constructor for the created Activity definition and set the Activity Definition properties.
- CelloSaaS.WorkFlow.Model.DIITaskActivityDefinition Properties
 - FullyQualifiedType – Activity Type Name.
 - Assembly – Assembly name of the Activity.
- CelloSaaS.WorkFlow.Model.SPTaskActivityDefinition Properties
 - FullyQualifiedType – Activity Type Name.
 - Assembly – Assembly name of the Activity.
 - SpDefinition – CelloSaaS.WorkFlow.Model.SPDefinition details.
 - ConnectionStringName – SP Connection string name.
 - SpName – Name of the SP.
 - ConnectionString – SP connection string.
 - Execution Type – Enum value(ExecuteScalar, ExecuteReader and ExecuteNonQuery)
 - InputMappings – CelloSaaS.Model.WorkFlow.SpInputMapping details. This contain ParameterName and ParameterDirection.

Sample for DIITaskActivityDefinition

```

/// <summary>
/// This class is used to define the Sending Email Activity
/// </summary>
public class SendingEmailDLLActivityDefinition : DLLTaskActivityDefinition
{
    /// <summary>
    /// This constructor is used to set the DLL Task Activity Definition details
    /// </summary>
    public SendingEmailDLLActivityDefinition()
    {
        FullyQualifiedType = typeof(SendingEmailActivity).FullName;
        Assembly = typeof(SendingEmailActivity).Assembly.GetName().Name;
    }
}

```

Sample for SPTaskActivityDefinition

```

/// <summary>
/// This class is used to define the Sending Email Activity
/// </summary>
public class SendingEmailSPActivityDefinition : SPTaskActivityDefinition
{
    /// <summary>
    /// This constructor is used to set the SP Task Activity Definition details
    /// </summary>
    public SendingEmailSPActivityDefinition()
    {
        FullyQualifiedType = typeof(SendingEmailActivity).FullName;
        Assembly = typeof(SendingEmailActivity).Assembly.GetName().Name;
        SpDefinition = new SPDefinition()
        {
            ConnectionStringName = "CelloSaaSConnectionString",
            ExecutionType = ExecuteType.ExecuteNonQuery,
            SPName = "GetEmailDetails",
            InputMappings = new List<SpInputMapping>()
            {
                new SpInputMapping()
                {
                    ParameterName = "Id",
                    Direction= System.Data.ParameterDirection.Input
                },
                new SpInputMapping()
                {
                    ParameterName = "EmailId",
                    Direction= System.Data.ParameterDirection.Output
                }
            };
        };
    }
}

```

Register the Activity Definition

- After the activity and activity definition created, need to register the Activity against the Workflow.
- Same activity can be registered in multiple workflows.
- Activity Table contains the following details, Type – Dll or StoredProcedure

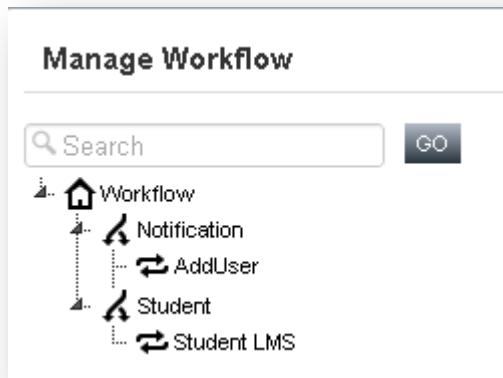
Column	Data Type	Allow Nulls
ActivityId	Uniqueidentifier	No
Name	nvarchar(200)	No
Type	Varchar(50)	No
Description	nVarchar(MAX)	No
WorkflowId	Uniqueidentifier	No
TypeName	Varchar(255)	No
AssemblyName	Varchar(255)	No
CreatedBy	Varchar(50)	No
CreatedOn	Datetime	No
UpdatedBy	Varchar(50)	Yes
UpdatedOn	Datetime	Yes
Status	Bit	No
TenantId	Uniquedentifier	Yes
Outcomes	nVarchar(MAX)	Yes

Example

Field	Value
ActivityId	cee5c059-e674-4816-ab71-8cb310d6a14d
Name	SendingMailActivity
Type	Dll
Description	SendingEmail Activity.
WorkflowId	4ee5c059-e674-4816-ab71-8cb310d6a14d
TypeName	CelloSaaS.WorkFlow.ActivityDefinition.SendingEmailAllActivityDefinition
AssemblyName	CelloSaaS.WorkFlow
CreatedBy	3398f837-b988-4708-999d-d3dfe11875b3
CreatedOn	2013-05-31
Status	TRUE
Outcomes	EmailSend, Completed

Field	Value
ActivityId	cee5c059-e674-4816-ab71-8cb310d6a14d
Name	SendingMailActivity
Type	Dll
Description	SendingEmail Activity.
WorkflowId	4ee5c059-e674-4816-ab71-8cb310d6a14d
TypeName	CelloSaaS.WorkFlow.ActivityDefinition.SendingEmailAllActivityDefinition
AssemblyName	CelloSaaS.WorkFlow
CreatedBy	3398f837-b988-4708-999d-d3dfe11875b3
CreatedOn	2013-05-31
Status	TRUE
Outcomes	EmailSend, Completed

Manage Workflows



Edit Workflow Design

User can edit the workflow upon click the edit icon, the workflow details with workflow name, description, and active version are displayed. It has three different designs like Default Workflow Design, Published Workflow Design, and Un-Published Workflow Design.

- The Default Workflow Design contains version, published status, date created and updated. The Product Admin can only have the permissions to edit the default workflow design.
- The Published Workflow Design contains the list of published workflows. Through this the user can view the workflow design. Using copy option the tenant can copy the workflow and change it for own. And only one published workflow will be in Active status.
- The Un-Published Workflow Design lists out the workflows which are unpublished.

Following API's are used to get the Workflow Designs.

```

/// <summary>
/// Gets the default workflow definition based on the workflow Id and filter Id.
/// </summary>
/// <param name="workflowId">The workflow id.</param>
/// <param name="filterId">The filter id.</param>
/// <returns></returns>
WorkflowDefinition GetDefaultDefinition(string workflowId, string filterId)

/// <summary>
/// Gets the published workflow definition details for the tenant and filter.
/// </summary>
/// <param name="workflowId">The workflow id.</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="filterId">The filter id.</param>
/// <returns></returns>
WorkflowDefinition GetWorkflowDefinition(string workflowId, string tenantId, string filterId)

/// <summary>
/// Gets all workflow definitions from own settings for the tenant.
/// </summary>

```



```
/// <param name="workflowId">The workflow id.</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="filterId">The filter id.</param>
/// <returns></returns>
List<WorkflowDefinition> GetAllWorkflowDefinitionsFromOwnSetting(string workflowId, string tenantId, string filterId)

/// <summary>
/// Gets all workflow definitions.
/// </summary>
/// <param name="workflowId">The workflow id.</param>
/// <param name="tenantId">The tenant id.</param>
/// <param name="filterId">The filter id.</param>
/// <returns></returns>
List<WorkflowDefinition> GetAllWorkflowDefinitions(string workflowId, string tenantId, string filterId)
WorkFlow Runtime

- Following API's used to create the workflow instance.
- These API's used to create the instance based on the tenant id, workflow id, map Id(Application specific unique value), filter id (application specific) and repository(batch or not).
- This will return the CelloSaaS.WorkFlow.WorkflowInstanceService.
- Using this service, can start, complete, pause and resume the workflow

```

Name Space: CelloSaaS.WorkFlow

Class: WorkflowInstanceStateService

```
/// <summary>
/// This method is used to Creates the instance for the given map Id against the workflow and tenant.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <param name="workflowId">The workflow id.</param>
/// <param name="mapId">Map id (Application Related Identifier)</param>
/// <param name="input">Workflow input details</param>
/// <param name="input">User Identifier</param>
/// <returns></returns>
WorkflowInstanceStateService CreateInstance(string tenantId, string workflowId, string mapId, IWorkflowInput input, string userId)
/// <summary>
/// This method is used to Creates the instance for the given map Id against the workflow and tenant.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <param name="workflowId">The workflow id.</param>
/// <param name="mapId">Map id (Application Related Identifier)</param>
/// <param name="input">Workflow input details</param>
/// <param name="input">User Identifier</param>
/// <param name="batchRepository">Batch repository</param>
/// <returns></returns>
WorkflowInstanceStateService CreateInstance(string tenantId, string workflowId, string mapId, IWorkflowInput input, BatchRepository batchRepository, string userId)
/// <summary>
/// This method is used to Creates the instance for the given map Id and filter Id against the workflow and tenant.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <param name="workflowId">The workflow id.</param>
/// <param name="mapId">Map id (Application Related Identifier)</param>
/// <param name="input">Workflow input details</param>
/// <param name="input">User Identifier</param>
/// <param name="batchRepository">Batch repository</param>
```



```
/// <param name="filterId">Filter id (Application Specific Id) </param>
/// <returns></returns>
WorkflowInstanceService CreateInstance(string tenantId, string workflowId, string mapId,
IWorkflowInput input, BatchRepository batchRepository, string userId, string filterId)
    • Following API's used to get the existing workflow instance services.
    • These methods are used to start, complete, pause and resume the Manual Task instance.

Name Space: CelloSaaS.WorkFlow
Class: WorkflowInstanceService
/// <summary>
/// This method is used to get the workflow instance service for the instance id.
/// </summary>
/// <param name="instanceId">workflow instance id.</param>
/// <param name="input">workflow input.</param>
/// <returns></returns>
WorkflowInstanceService GetInstance(Guid instanceId, IWorkflowInput input)

/// <summary>
/// This method is used to get the workflow instance service for the instance id.
/// </summary>
/// <param name="instanceId">workflow instance id.</param>
/// <param name="input">workflow input.</param>
/// <param name="batchRepository">The batch repository.</param>
/// <returns></returns>
WorkflowInstanceService GetInstance(Guid instanceId, IWorkflowInput input, BatchRepository
batchRepository)

/// <summary>
/// This method is used to get the workflow instance service for the map id.
/// </summary>
/// <param name="mapId">map id</param>
/// <param name="input">workflow input details</param>
/// <returns></returns>
WorkflowInstanceService GetInstance(string mapId, IWorkflowInput input)

/// <summary>
/// This method is used to get the workflow instance service for the map id.
/// </summary>
/// <param name="mapId">map id</param>
/// <param name="input">workflow input details</param>
/// <param name="batchRepository">The batch repository.</param>
/// <returns></returns>
WorkflowInstanceService GetInstance(string mapId, IWorkflowInput input, BatchRepository
batchRepository)

/// <summary>
/// This method is used to get the workflow instance service for the map id and the workflow id.
/// </summary>
/// <param name="mapId">The map id.</param>
/// <param name="wfId">The wf id.</param>
/// <param name="input">The input.</param>
/// <returns>The Workflow Instance Service</returns>
/// <see cref="WorkflowInstanceService"/>
WorkflowInstanceService GetInstance(string mapId, string wfId, IWorkflowInput input)

/// <summary>
/// This method is used to get the workflow instance service for the map id and the workflow id.
/// </summary>
```



```
/// <param name="mapId">The map id.</param>
/// <param name="wfId">The wf id.</param>
/// <param name="input">The input.</param>
/// <param name="batchRepository">The batch repository.</param>
/// <returns>The Workflow Instance Service</returns>
/// <see cref="WorkflowInstanceService"/>
WorkflowInstanceService GetInstance(string mapId, string wfId, IWorkflowInput input,
BatchRepository batchRepository)

/// <summary>
/// This method is used to get the workflow instance service for the workflow instance
/// </summary>
/// <param name="wfInstance">The wf instance.</param>
/// <param name="input">The input.</param>
/// <param name="batchRepository">The batch repository.</param>
/// <returns></returns>
WorkflowInstanceService GetInstance(WorkflowInstance wfInstance, IWorkflowInput input,
BatchRepository batchRepository)

/// <summary>
/// This method is used to get the workflow instance service for the workflow instance
/// </summary>
/// <param name="wfInstance">The wf instance.</param>
/// <param name="input">The input.</param>
/// <returns></returns>
WorkflowInstanceService GetInstance(WorkflowInstance wfInstance, IWorkflowInput input)
```

Workflow input

- Create class that inherits from the CelloSaaS.Workflow.Model.IWorkflowInput.
- Use the workflow input model to get and create the workflow instance service.

Example:

```
/// <summary>
/// Workflow Input class
/// </summary>
public class BatchInput : IWorkflowInput
{
    public string FilterId { get; set; }
    public string BatchId { get; set; }
}
```

- Start the workflow using the following method in WorkflowInstanceService.

```
/// <summary>
/// Starts the workflow instance.
/// </summary>
public void Start()
```

Example:

```
var input = new BatchInput()
{
    TenantId = UserIdentity.TenantID,
    FilterId = "1",
    BatchId = "1200"
};
var instanceService = WorkflowInstanceService.CreateInstance(input.TenantId, "4EE5C059-E674-
4816-AB71-8CB310D6A14D",input.BatchId, input, null, UserIdentity.UserId, input.filterId);
instanceService.Start();
```

- Following API's used to get the WorkflowTaskInstanceService from the WorkflowInstance Service.

```

/// <summary>
/// Gets the active task instance service for task code and task actor.
/// </summary>
/// <param name="taskCode">task code</param>
/// <param name="taskActor">task actor</param>
/// <returns></returns>
WorkflowTaskInstanceService GetActiveTaskInstanceService(string taskCode, ITaskActor taskActor)
/// <summary>
/// Gets the active task instance services for task actor.
/// </summary>
/// <param name="taskActor">The task actor.</param>
/// <returns></returns>
List<WorkflowTaskInstanceService> GetActiveTaskInstanceService(ITaskActor taskActor)

/// <summary>
/// Gets the active task instance services for task code
/// </summary>
/// <param name="taskCode">task code</param>
/// <returns></returns>
List<WorkflowTaskInstanceService> GetActiveTaskInstanceService(string taskCode)
/// <summary>
/// Gets the active task instance service for task code and task current execution status.
/// </summary>
/// <param name="taskCode">task code</param>
/// <param name="status">task current execution status</param>
/// <returns></returns>
WorkflowTaskInstanceService GetTaskInstance(string taskCode, string status)
    • Workflow task instance service is used to start, execute and complete the particular task.
    • Following API's available in workflow task instance service.

/// <summary>
/// Starts the task.
/// </summary>
/// <param name="objects">Workflow inputs</param>
public void StartTask(params object[] objects)

/// <summary>
/// Executes the task.
/// </summary>
/// <param name="objects">Workflow inputs</param>
public void ExecuteTask(params object[] inputObjects)
/// <summary>
/// Completes the task.
/// </summary>
/// <param name="activity">The activity</param>
public void CompleteTask(IActivity activity)

```

Workflow service

- Workflow service is used to search workflow instance, workflow task instance, get active task instance for an actor and get map ids for an actor.

```

/// <summary>
/// This method is used to get the workflow instance details based on the search conditions
/// </summary>
/// <param name="wfSearchParameter">Workflow search conditions</param>

```



```
/// <returns></returns>
List<WorkflowInstance> SearchWorkFlowInstance(WfSearchParameter wfSearchParameter)

/// <summary>
/// This method is used to get the work flow instance with task instance details based on the search
conditions
/// </summary>
/// <param name="wfSearchParameter">Workflow search conditions</param>
/// <returns>List of WorkflowInstance</returns>
List<WorkflowInstance> SearchWorkFlowInstanceDeep(WfSearchParameter wfSearchParameter)

/// <summary>
/// This method is used to get the workflow task instance details based on the search conditions
/// </summary>
/// <param name="wfSearchParameter">Workflow search conditions</param>
/// <returns></returns>
public      List<WorkflowTaskInstance>      SearchWorkFlowTaskInstance(WfSearchParameter
wfSearchParameter)

/// <summary>
/// This method is used to get the active task instance for the Actor.
/// This will return the task has the started, executed and errored status.
/// </summary>
/// <param name="actor">The task actor.</param>
/// <returns></returns>
List<WorkflowTaskInstance> GetActiveTaskInstanceForActor(ITaskActor actor)

/// <summary>
/// This method is used to get the active task instance for the Actor and the given task code.
/// This will return the task has the started, executed and errored status.
/// </summary>
/// <param name="taskCode">The task code.</param>
/// <param name="actor">The task actor.</param>
/// <returns></returns>
List<WorkflowTaskInstance> GetActiveTaskInstanceForActor(string taskCode, ITaskActor actor)

/// <summary>
/// This method is used to get the map ids for the Actor and the given task code.
/// This will return the task has the started, executed and errored status.
/// </summary>
/// <param name="actor">The task actor.</param>
/// <returns></returns>
List<string> GetMapIdsForActor(ITaskActor actor)

/// <summary>
/// This method is used to get the map ids for the Actor and the given task code.
/// This will return the task has the started, executed and errored status.
/// </summary>
/// <param name="taskCode">The task code.</param>
/// <param name="actor">The task actor.</param>
/// <returns></returns>
List<string> GetMapIdsForActor(string taskCode, ITaskActor actor)

/// <summary>
/// This method is used to restart the particular errored step in the workflow.
/// </summary>
/// <param name="wfId">The wf id.</param>
```

```

/// <param name="wfInstanceId">The wf instance id.</param>
/// <param name="mapId">The map id.</param>
/// <param name="taskCode">The task code.</param>
public bool RestartWorkflowStep(string wfId, string wfInstanceId, string mapId, string taskCode,
string tenantId)
Batch Workflow Service

- This service is used to create and execute the batch workflow.
- APIs used to create, get and commit the workflow as batch.


/// <summary>
/// Creates the instance using batch repository.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <param name="workflowId">The workflow id.</param>
/// <param name="mapId">The map id.</param>
/// <param name="input">The input.</param>
/// <param name="userId">The user id.</param>
/// <returns></returns>
WorkflowInstanceService CreateInstance(string tenantId, string workflowId, string mapId,
IWorkflowInput input, string userId)

/// <summary>
/// Creates the instance using batch repository.
/// </summary>
/// <param name="tenantId">The tenant id.</param>
/// <param name="workflowId">The workflow id.</param>
/// <param name="mapId">The map id.</param>
/// <param name="input">The input.</param>
/// <param name="userId">The user id.</param>
/// <returns></returns>
WorkflowInstanceService CreateInstance(string tenantId, string workflowId, string mapId,
IWorkflowInput input, string userId, string filterId)

/// <summary>
/// Gets the workflow instance service for the instance ids.
/// </summary>
/// <param name="inputWithInstanceId">Instance Ids with Workflow inputs</param>
/// <returns></returns>
List<WorkflowInstanceService> GetInstances(Dictionary<Guid,
inputWithInstanceId>, IWorkflowInput>

/// <summary>
/// Gets the workflow instance service for the instance ids.
/// </summary>
/// <param name="inputWithMapId">Map Ids with Workflow inputs</param>
/// <returns></returns>
List<WorkflowInstanceService> GetInstances(Dictionary<string, IWorkflowInput> inputWithMapId)
/// <summary>
/// Commits the specified batch repository. This will save all instance, task instance, outcome and
execution status.
/// </summary>
void Commit()

```

Workflow Task Actor Service

- This service is used to add/update/delete the dynamic actor details for the task.
- Get the actor details for the particular task.
- Dynamic task actor details will be stored in the WFDynamicTaskActor table once task is started then actor details will be deleted from the WFDynamicTaskActor table and stored in the WFTaskActor table.

- Actor details only applicable for the Manual Task.

```

/// <summary>
/// Gets the task actors details based on the actor definition and tenant id.
/// </summary>
/// <param name="actorDefinition">The actor definition.</param>
/// <returns>List of ITaskActor</returns>
/// <see cref="ITaskActor"/>
List<ITaskActor> GetTaskActors(ITaskActorDefinition actorDefinition, string tenantId)

/// <summary>
/// Gets the task actor details for the particular task.
/// </summary>
/// <param name="taskId">Task id</param>
/// <param name="workflowId">Workflow id</param>
/// <param name="tenantId">Tenant id</param>
/// <returns>List of ITaskActor</returns>
List<ITaskActor> GetTaskActors(string taskId, string workflowId, string tenantId)

/// <summary>
/// Gets the dynamic task actor ids.
/// </summary>
/// <param name="wfInstanceId">The wf instance id [Mandatory].</param>
/// <param name="taskCode">The task code [Mandatory].</param>
/// <returns>List of ITaskActor</returns>
/// <see cref="ITaskActor"/>
List<ITaskActor> GetDynamicTaskActors(string wfInstanceId, string taskCode)

/// <summary>
/// This method is used to delete all dynamic task actors for the workflow instance id and task code.
/// </summary>
/// <param name="wfInstanceId">The wf instance id [Mandatory].</param>
/// <param name="taskCode">The task code [Mandatory].</param>
void DeleteDynamicTaskActors(string wfInstanceId, string taskCode)

/// <summary>
/// This method is used to delete the dynamic task actors for the workflow instance id and task code.
/// </summary>
/// <param name="wfInstanceId">The wf instance id [Mandatory].</param>
/// <param name="taskCode">The task code [Mandatory].</param>
/// <param name="actorIds">The actor ids.</param>
void DeleteDynamicTaskActors(string wfInstanceId, string taskCode, string[] actorIds)

/// <summary>
/// This method is used to updates the dynamic task actor details for mapId.
/// </summary>
/// <param name="mapId">The map id [Mandatory].</param>
/// <param name="taskActorList">The task actor list [Mandatory].</param>
/// <param name="tenantId">The tenant id [Mandatory].</param>
void UpdateDynamicTaskActorDetails(string mapId, Dictionary<string, List<string>> taskActorList,
string tenantId)

/// <summary>
/// This method is used to updates the dynamic task actor details for workflow instance id.
/// </summary>
/// <param name="wfInstanceId">The wf instance id [Mandatory].</param>
/// <param name="taskActorList">The task actor list [Mandatory].</param>
/// <param name="tenantId">The tenant id [Mandatory].</param>

```

```

void UpdateDynamicTaskDetailsByWfInstanceId(string wfInstanceId, Dictionary<string,
List<string>> taskActorList, string tenantId)

/// <summary>
/// This method is used to delete the actor details for the tasks.
/// </summary>
/// <param name="actorIdWithTaskInstanceId">The actor id with task instance id.</param>
void DeleteTaskActorDetails(Dictionary<Guid, string> actorIdWithTaskInstanceId)

```

Workflow Tracking Service [Task Monitoring]

- If a task has expiry condition it is monitored
- A task is also monitored if the complete condition is not getting completed in an automatic/manual task or start condition is not started in a manual /automatic task.
- Workflow tracking window service is used to do the task monitoring.

Batch Workflow Windows Service

- Batch workflow service is used to start and execute the long running workflow tasks.
- Need to put workflow instance entry in Workflow request table using Workflow request service.

Sample API Usage

How to start the Workflow

- To start workflow uses the following API. For more details please for Developer Guidance Section under **WorkFlow Runtime**.
- `WorkflowInstanceService CreateInstance(string tenantId, string workflowId, string mapId, IWorkflowInput input, string userId)`
- Example Code:

```

WorkflowInstanceService instanceService =
WorkflowInstanceService.CreateInstance(input.TenantId, workflowId, mapId,
input, null, input.UserId, filterId);
instanceService.Start();

```

- You can pass your input through input class which is inherited from CelloSaaS.WorkFlow.Model.IWorkflowInput.

How to get and execute the Manual Task

- You can get the WorkFlowInstanceService from the following API. For more details please for Developer Guidance Section under **WorkFlow Runtime**.
- `WorkflowInstanceService GetInstance(Guid instanceId, IWorkflowInput input)`
- You can get the WorkFlowInstanceService using instance id or Map Id(your application related unique Id).
- Using WorkFlowInstanceService you can get the TaskInstance by passing taskCode and Actor Id.
- After got the WorkFlowTaskInstanceService, you can call ExecuteTask to execute the particular task.
- Example Code:

```

WorkflowInstanceService instanceService =
WorkflowInstanceService.GetInstance(Guid.Parse(instanceId), new BatchInput() {
BatchId = Guid.NewGuid().ToString(), CategoryId = "Test", IsComplete = true });

```

```

WorkflowTaskInstanceService taskInstanceService =
instanceService.GetActiveTaskInstanceService(taskCode, new TaskActor() {
UserId = UserIdentity.UserId });

```

```

taskInstanceService.ExecuteTask(new BatchInput() { IsComplete = status,
BatchId = Guid.NewGuid().ToString(), CategoryId = "Test" });

```

16 CELLOSAAS BUSINESS RULES ENGINE

A business rule engine (BRE) is a component of software allowing programmers or end users to change the business logic in any real world enterprise application. To carry out a business policy or procedure, a business rule or statement is required. Business logic uses data in a database and a sequence of operations to carry out the business rule. A business rules engine (BRE) separates business logic from your mission-critical applications in order to gain agility and improve operational performance. To get the most benefit from this application architecture, you need a business rules engine that:

Empowers business users to create and manage business rules with minimal involvement from IT staff.

Supports sophisticated, powerful rules that can capture your business workflow and your policies and procedures in all their dynamic complexity.

Integrates seamlessly with your existing IT assets and scales for enterprise-class performance. CelloSaaS provides a simplified Business Rules Engine [BRE], easy to configure and helps the developers to develop fairly complex rules required by most business applications. It makes your business logic transparent and flexible by providing an extensible environment where both business users/experts/developers can configure the business logic. It also allows customizing business rules at the tenant level, which is one of the most sought after features in a multi-tenant application.

Benefits of CelloSaaS Business Rule implementations:

Improved transparency and communication between users and IT community

Improved maintainability& agility: Leveraging Business Rules makes the process of changing business logic much easier, because change requests can be implemented without changes to program code by isolating the change, allowing to quickly deploy the rule changes into production and have them immediately accessible by applications to be executed at or near real-time.

Increased Flexibility: Business applications want to behave their application differently for different classes of customers or for individual customers. This may involve the same basic application that applies different rules based upon each customer's contract terms.

Reduced time-to-market

CelloSaaS BRE includes:

Business Rule Editor: This is an intuitive interface allowing business users to design, define, document and edit business rules.

Rules Engine Execution Core: This is a programming code enforcing the rules.

Business Rule Repository: This is a persistence layer storing business rules, which are defined by business users.

Reporting Component: This is an intuitive interface allowing business users to query and report existing rules.

16.1 What Role Does CelloSaaS BRE Play in Your Business?

Business rules are core features by which process engine work and it is this business rule set that separates one company from another. Business Rules are used to implement business policies. These are the policies used to run businesses as understood by the customers, employees, and partners.

The business rules can be used in the scenarios where the business logic needs to be changed in the course of application run-time or per tenant wise logic needs to be changed.

It is a rule that defines or constrains some aspect of business. Business rules are intended to assert, control or influence the behavior of the business. It describes the definitions, operations and the

constraints that apply to an organization. Below are some of the real world use cases where the BRs helps to solve the business problems, they are:

Case 1: The business rule may calculate discount, shipping cost for a customer order depending on order amount, festive season, customer type, etc., Business rule may calculate the evaluation of an object depending of various factors.

Your application will communicate with the rule engine by specifying the rules to execute and passing in the necessary data. Once the rules execute, your application will use the rules engine results to display to the end user or to execute another process.

Case 2: A HR manager of a tenant would like to customize the business rule with regards to benefits administration. A purchase manager would like to customize the workflow regarding vendor approval or Purchase Order request.

There are number of benefits that can be derived from the use of Business Rules. The three most important benefits are:

Agility: Simple and rapid response to changing requirements.

Cost reduction: Lower cost to create or update the parts of applications that implement business policies.

Transparency: Rules allow management to easily audit that software services implement their corresponding business policies.

CelloSaaS user interface for business rules is built on Silverlight to deliver rich user experiences and is complemented with workflow and business rules engines to deliver process automation and software intelligence capabilities.

CelloSaaS business rules engine is developed on the top of NXBRE. (NXBRE is a popular open source business rules engine distributed under LGPL). It allows each tenant to come up with their own business rule.

16.2 Crafting a Business Rule

Creating an accurate, complete, and comprehensible business rule takes certain skill, discipline. There are certain techniques to achieve the goal of creating business rule. The following are the steps on how a rule is created.

- ☞ Step 1: **Extract the business rule:** There are numerous sources for a business rule, so the first decision is to determine the best source for the set of business rules. Namely, SME's, Documents (both business and technical). CelloSaaS facilitates business rule capture through:
 - Intensive business rule capture with knowledge experts
 - Pragmatic, effective and proven approach
 - Facilitate divergent views and different personalities
- ☞ Step 2: Determine whether the business rule computes or derives something.
- ☞ Step 3: Start with the subject.
- ☞ Step 4: Choose the form of the business rule keyword.
- ☞ Step 5: Add the condition(s).
- ☞ Step 6: Make sure the business rules all fit together.
- ☞ Step 7: Construct a Rule Set

A rule set is a description of a set of rules to be executed. A rule set is a set of business rules that are executed to calculate and return some business values. You need to define a new rule set at this point. This is the core part of the exercise.

- ☞ Step 8: Specify the parameter for the Rule
- ☞ Step 9: Call the business rule at appropriate location (e.g. How do you trigger the business rule), etc.?

16.3 Components of Business Rules

The CelloSaaS framework allows each product to define its own domain model and map it to the internal classes and variables used in the BRE, so that the end user who is doing the customization can deal with "easily understandable" entities and activities. Similarly, the framework also provides complete control to the product developer to decide what aspects of the business rule should be standard and what aspects should be exposed for customization by end users. CelloSaaS consists of 3 components, namely:

- CelloSaaS.Rules (Rules Library)
- CelloSaaS.Silverlight.Rules (Silverlight Client application)
- CelloSaaS.WCF.Rules (WCF application)

16.4 Rules Library

Rule Library allow you to define all the conditions that reflect the organization's business rules that can be applied on any process defined by the organization, that is, a library rule contains the definition of a macro rule of the business. The Rule library contains the business rule engine and rule definition classes and APIs to define and execute the business rules.

Libraries that have to be referenced in your application project, they are:

- CelloSaaS.Rules.Core
- CelloSaaS.Rules.Constants
- CelloSaaS.Rules.Execution
- CelloSaaS.Rules.Utils

16.5 Rules Web Service Application

In a distributed environment/LOB applications, the rules that you created in your application might be usable in another application as well, so the BRs should be exposable to the outer world in order to avoid the effort duplication. The CelloSaaS rules services are exposed as WCF services for consuming from any application and from any environment, all the developers has to do is to host this application and change the appropriate connection strings.

16.6 Silverlight Client Application

16.6.1 Why Silverlight?

An integrated approach to authoring, storing, managing and executing rules can deliver consistent decisions, lower maintenance costs, and improved time to value for key applications. Silverlight offers a great visual experience for both designing and using applications. One of the major advantage of using Silverlight is it brings higher ease of use, better user engagement while defining the rules.

This application consists of UI to define/customize the business rules by the end user. The .XAP file can be found in the following folder ~/Silverlight/ of the web application. This application uses the WCF Rule service.

The silverlight UI consists of two types of screen for different types of rule.

- IfThenElseRule
- DecisionTableRule

initParams needs to be passed to the Silverlight UI as follows to correctly load the rules in the corresponding screen.

```
<param name="initParams"
value="TenantId=tenantId;ScreenType=DecisionTableRule;RulesetCode=TestRule;
WCFServiceURL=http://localhost/cellosaasruleservices"/>
```

```
<divid="silverlightControlHost" style="z-index: -1; height: 600px; width: 99.99%; border: 1px solid #ccc">
<% Uri uri = HttpContext.Current.Request.Url; string host = uri.Scheme + Uri.SchemeDelimiter +
uri.Host + ":" + uri.Port; string wcfUrl = host + "/RuleServices"; %>
<object data="data:application/x-silverlight-2" type="application/x-silverlight-2" width="100%" height="100%">
<param name="source" value="/Silverlight/CelloSaaS.BusinessRules.UI.xaml"/>
<param name="initParams" value="RuleSetCode=<%=ViewData["RulesetCode"]%>, TenantId=<%=ViewData["TenantId"]%>, ScreenType=<%=ViewData["ScreenType"]%>, WCFServiceURL=<%=wcfUrl%>"/>
<param name="onError" value="onSilverlightError"/>
<param name="background" value="white"/>
<param name="minRuntimeVersion" value="4.0.50826.0"/>
<param name="autoUpgrade" value="true"/>
<a href="http://go.microsoft.com/fwlink/?LinkId=149156&v=4.0.50826.0" style="text-decoration: none">

</a>
</object>
<iframe id="_sl_historyFrame" style="visibility: hidden; height: 0px; width: 0px; border: 0px"></iframe>
</div>
```

You can construct this information in the controller and pass it via View Data. Use the below lines of code to bring up the Silverlight UI in any page as you like.

CelloSaaS Business Rule contains the following

RuleSet

It contains the user configured rule condition/statements ready for execution in the form of XML persisted in the database. It is defined by the user at runtime through a Silverlight UI.

Activities

It provides the variables for defining the rule and its values upon execution. Activities are predefined at design time or can be dynamic by using Xml configuration.

Activities are of following types:

Entity Rule Activity

You can register a Business entity to this activity and it automatically registers all the entity properties and its extended fields as rule variables for configuration.

Open Rule Activity

Developer has to write some classes and register it. Here the developer has complete controls over which variables which are participating in the business rule.

Xml Rule Activity

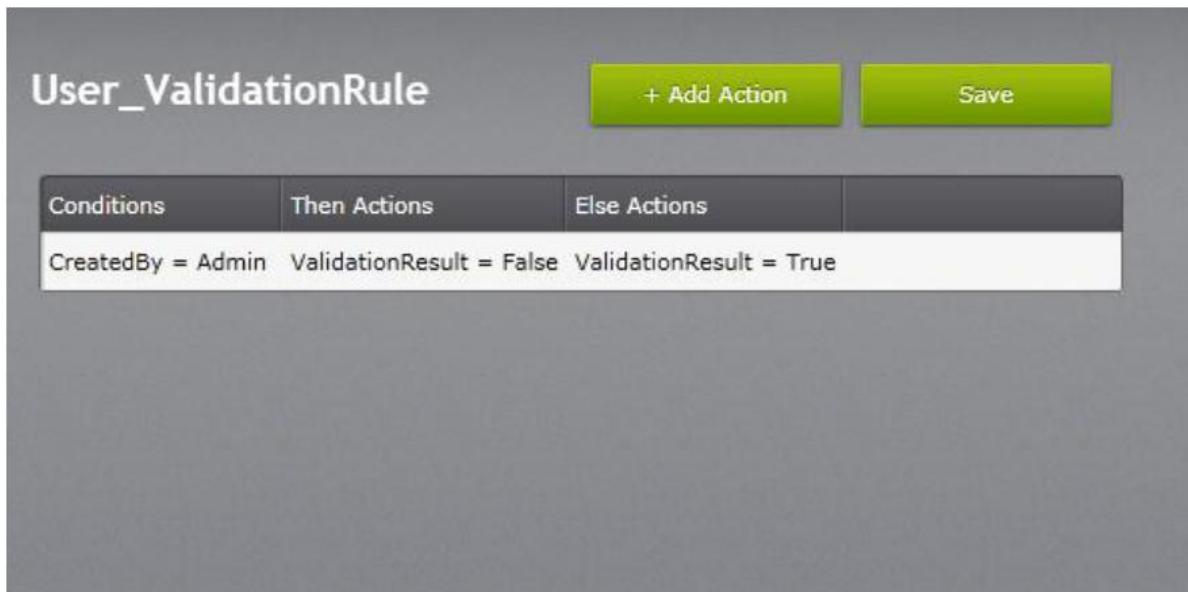
Here the variables are registered in the form of XML. Mostly used when defining dynamic business rules.

Business Rule Designer Types:

- If Then Else Condition
- Decision Table

If Then Else Condition

If the conditions evaluated are true then the statements given in the “Then” section will be evaluated else the statements in the “Else” part will be evaluated. Here the tenant can change conditions and actions as they want using simple logical and relational operators.



The screenshot shows a user interface for defining a validation rule. At the top, there's a title bar with the text "User_ValidationRule". To the right of the title are two buttons: "+ Add Action" and "Save". Below the title bar is a horizontal navigation bar with three tabs: "Conditions", "Then Actions", and "Else Actions". The "Conditions" tab is currently selected, showing the condition "CreatedBy = Admin". The "Then Actions" tab shows the action "ValidationResult = False". The "Else Actions" tab shows the action "ValidationResult = True".

[User Validation Rule- If then else rule]

User_ValidationRule

Cancel
Save

IF

CreatedBy	=	Admin	And
UserId	nResult	=	False
MembershipId	nResult	=	True
FirstName	nResult	=	
LastName	nResult	=	
AddressId	nResult	=	
CreatedOn	nResult	=	
CreatedBy	nResult	=	
UpdatedOn	nResult	=	
UpdatedBy	nResult	=	
Status	nResult	=	
User_Description	nResult	=	
Extn_PostBox	nResult	=	
ValidationResult	nResult	=	

[If-then-else Rule (Selecting Variables)]

Decision Table

This is like a table consisting of columns. The columns are of two types:

ConditionColumn
ActionColumn

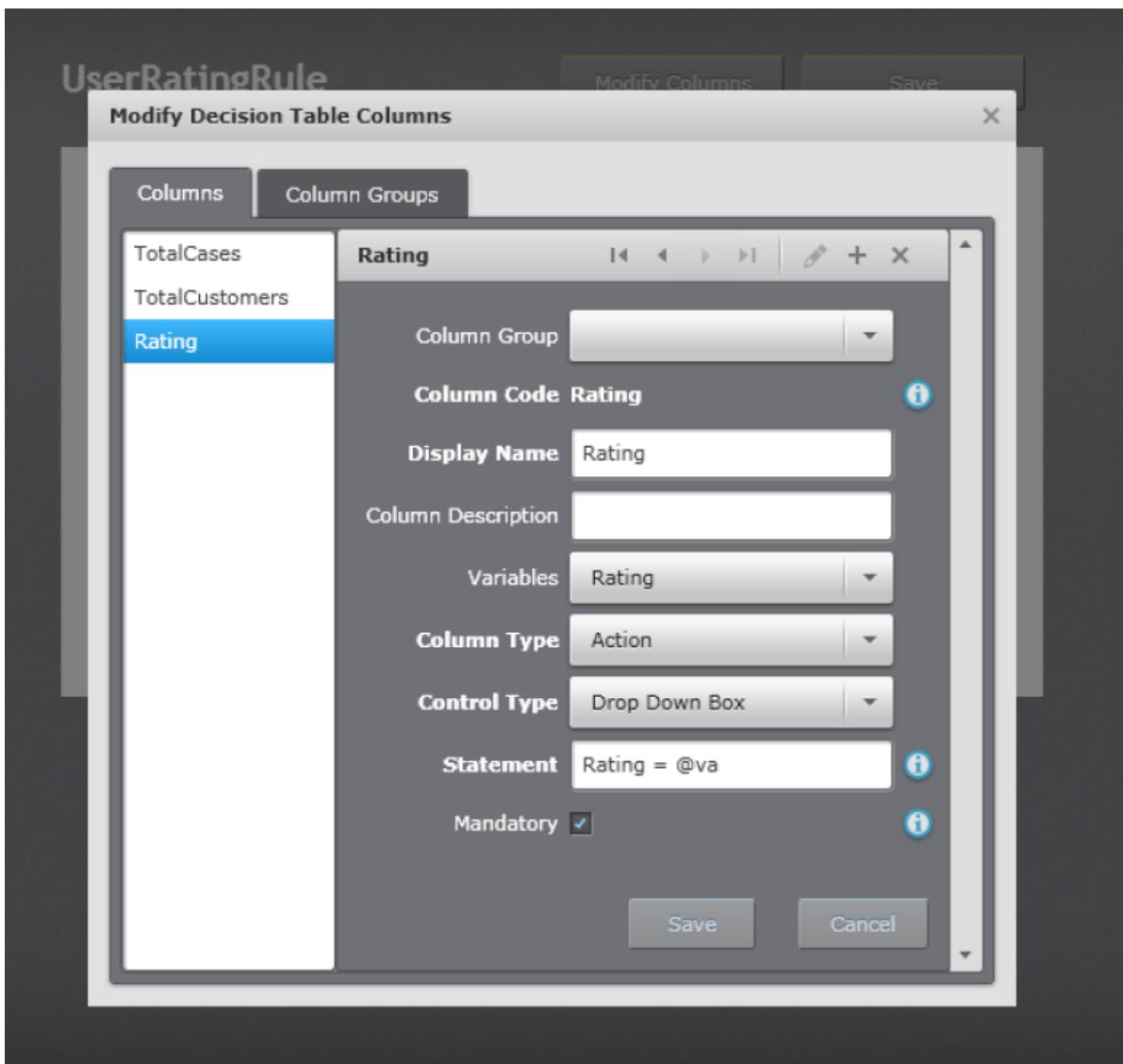
The condition columns of each Row are evaluated in order and if it is evaluated to true then the action columns are evaluated. If the conditions in the row are evaluated as false it begins evaluating the next row.

UserRatingRule

[Modify Columns](#)[Save](#)

Total Cases	Total Customers	Rating
1000	200	A
500	100	B
250	50	C
100	10	D
50	1	E
0	0	

[Decistion Table Business Rule- Sample]



[Manage Decision Table Business Rule]

Types of Business Rules:

Static Business Rules
 Dynamic Business Rules

Static Business Rules

These rules are identified during design time itself and developed and configured by the developer. Ie when we already know the business entities or variables which are participating in the business scenario we go for static business rules. At runtime the users can only change the configuration without affecting the variables/acting business entities. Also these static business rules are global to all tenants and more coupled with the application domain.

Dynamic Business Rules

In scenarios where we use more generic approach to solve business problems; and where we are not sure about the exact business entities or variables participating in the rules. In such cases you can use the Dynamic business rule works based on XML inputs. So the developer only need to



provision a mechanism to execute the Generic XML based rule and use its output into his business logic. These dynamic rules can be created at runtime and are specific to the tenant.

Rule Activity Mappings

Static Business rules must be designed and registered to CelloSaaS Business Rule Engine. This is one time registration process and should be done at application start event in the **global.asax**.

The below section describes on Creating a sample Rule and creating various Activities which are relevant to the rule.

Example Scenario:

Consider a call center operation where the call center agent manages the customer cases reported via phone, email, etc. The employee must be rated depending on how many customers and cases he/she has handled during the fiscal year. Rating may also depend on other parameters like customer country, experience, etc. Let's first define the variables required to construct the conditions.

Variables:

- Employee entity properties
- Total Customers Handled
- Total Cases Handled
- Rating – Output variable

Defining a Business Rule:

Write the below class which describes about the **RuleSetCode**, **Description**, **ScreenType**.

```
[RuleSet("User Rating Rule", ScreenType.DecisionTable, RuleSetCode = "UserRatingRule", Description = "User rating rule")]
```

```
public class UserRatingRule : CelloRuleBase
{
    public UserRatingRule(RuleSandbox sb, params IRuleInput[] inputVars)
        : base(sb, inputVars)
    {
    }

    public string GetRating()
    {
        if(vars.ContainsKey("Rating"))
            return vars["Rating"].ToString();
        return string.Empty;
    }
}

public class UserRatingRuleInput : IRuleInput
{
    [Expand]
    public UserDetails Entity { get; set; }

    public int TotalCases { get; set; }

    public int TotalCustomers { get; set; }
}
```

```

}

public class UserRatingRuleOutput : IRuleResult
{
    public RatingType Rating { get; set; }
}

public enum RatingType
{
    Exceptional,
    Excellent,
    Value,
    Performer,
    Newbie
}

```

Defining an Open Rule Activity:

Create a class deriving from `ActivityBase` class. This class registers the rule variables and populates variable value during execution.

```

public class RatingActivity : ActivityBase
{
    private const string RuleSetCode = "UserRatingRule";

    public override IRuleActivity CreateInstance()
    {
        return new RatingActivity();
    }

    public RatingActivity()
    {
        Name = "Rating Activity";
    }

    /// <summary>
    /// Fetchs the required details from database and populate
    /// the variables necessary the execute the rules
    /// </summary>
    /// <param name="vars">The vars.</param>
    protected override void PopulateFacts(IWorkingMemory vars)
    {
        if (!vars.ContainsKey("Entity")) return;
        var entity = vars["Entity"] as UserDetails;

        if (entity != null)
        {
            PopulateVarsFromModel(vars, entity);
            PopulateFactsForExtendedFields(entity);
        }
    }

    /// <summary>
    /// Registers the supported facts.
    /// </summary>
    /// <param name="variableRegistry">The variable registry.</param>
    protected override void RegisterSupportedFacts(IVariableRegistry variableRegistry)

```



```
{  
    variableRegistry.RegisterType(typeof(UserRatingRuleInput), RuleSetCode, this);  
    variableRegistry.RegisterType(typeof(UserRatingRuleOutput), RuleSetCode, this);  
  
    RegisterExtendedFields(variableRegistry, "UserDetails", RuleSetCode);  
}  
}
```

Registering and Initializing CelloSaaS Business Rule Engine:

Write a class which derives from `CelloSaaS.Rules.Execution.IRuleConfigurator` and use the Fluent API to register your business rules.

```
public class ApplicationBusinessRuleConfig : CelloSaaS.Rules.Execution.IRuleConfigurator  
{  
    public void Configure(CelloSaaS.Rules.Execution.RuleConfig configObj)  
    {  
        // mapping rule with open rule activities  
        configObj.RuleActivityMappings  
            .Add<UserRatingRule>()  
            .WithActivity<RatingActivity>();  
        // registering business entity rules  
        configObj.EntityRules.Add<Customer>()  
            .WithPreprocessorRule()  
            .WithValidationRule()  
            .Add<CaseDetail>()  
            .WithPreprocessorRule()  
            .WithValidationRule();  
  
    }  
}
```

In Global.asax `Application_Start` method, register it with the CelloSaaS,

```
RuleConfiguration.Register<ApplicationBusinessRuleConfig>();  
RuleConfiguration.Configure(); // initializes the CelloSaaS Business rule engine
```

To Executing a Business Rule

Below code shows you how to execute a static business rule.

```
// first get the RuleSandbox  
var ruleSandbox = CelloSaaS.Rules.Execution.Rules.GetSandbox(UserIdentity.TenantID);  
  
var rule = new UserRatingRule(ruleSandbox, new UserRatingRuleInput  
{  
    Entity = userDetails,  
    TotalCases = totalCases,  
    TotalCustomers = totalCustomers  
});  
  
rule.Execute(); // execute the rule  
  
string rating = rule.GetRating(); // get the output
```



Entity Validation and pre-processor Rules

CelloSaaS also provides you out of box business entity validation and pre-processing though business rules. The developer only need to register the business entity which requires dynamic tenant based validation/pre-preprocessing.

```
// registering business entity rules
configObj.EntityRules.Add<Customer>()
    .WithPreprocessorRule()
    .WithValidationRule()
.Add<CaseDetail>()
    .WithPreprocessorRule()
    .WithValidationRule();
```

Here `Customer` and `CaseDetail` are business entities which require tenant based dynamic validation/pre-processing before persisting it in the database. So when ever `DoCreate` or `DoUpdate` methods of these business entities DAL are called CelloSaaS automatically executes the business rules and throws exception if the validation fails.

For Entity validation rule, the below two variables are automatically registered.

`ValidationResult (bool)`

If set true then the `DoCreate/DoUpdate` method continues execution else `BusinessRuleException` will be thrown containing the “`ValidationErrorMessage`” string value.

`ValidationErrorMessage (string)`

This contains the validation error message. To separate multiple property error messages, use “`^#^`” (without quotes) as the delimiter.

How To Guide: [Implementing Custom Entity Validation](#)

Defining and executing a dynamic business rule:

Use the Business Rule UI to create a Business Rule as shown in the below screen shot.

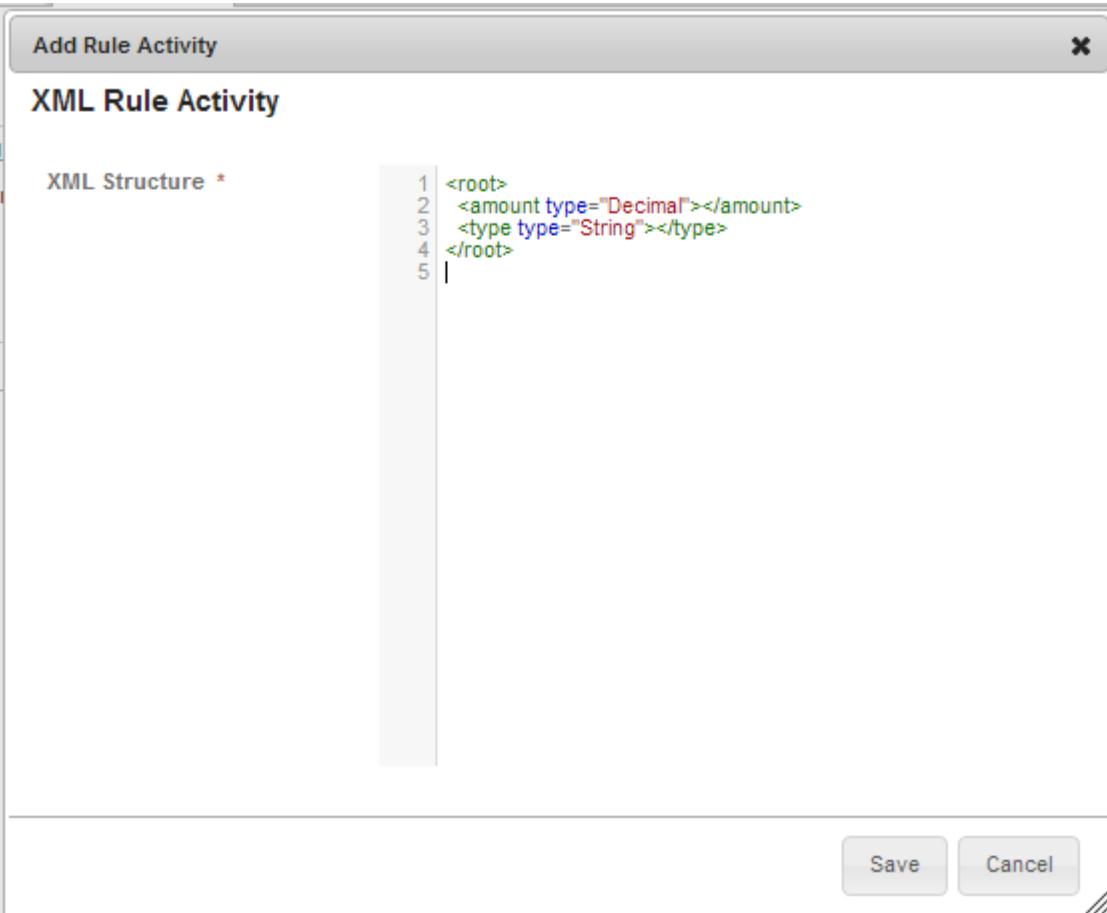
Create New Ruleset

RuleSet Code *	<input type="text"/>
RuleSet Name *	<input type="text"/>
Category	<input type="text"/>
Description	<input type="text"/>
Screen Type *	DecisionTable <input type="button" value="▼"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

[To create a new dynamic business rule]

Then add activities to the rule.

Rule Details	Rule Activities
	<input type="button" value="XML Rule Activity"/> <input type="button" value="Open Rule Activity"/> <input type="button" value="Entity Activity"/>
<u>XMLRuleActivity</u> <div style="border: 1px solid black; padding: 5px;"> XML Structure: <pre><root> <amount type="Decimal" ></amount> <type type="String" ></type> </root></pre> </div>	



[To Add XML Rule Activity]

An Xml Rule Activity requires a valid XML. The XML elements will be registered as rule Variables.

E.g:

```

<root>
  <amount type="Decimal"></amount>
  <type type="String"></type>
</root>

```

Here “amount”, “type” will be registered as rule variable which will be available in the Silverlight Configuration UI.

Supported data types:

string, int, decimal, date, bool.

When adding an Xml Rule Activity, a variable named “Output” of type sting is automatically registered. You can use this variable to hold the result.

To execute a dynamic rule

```
var outputXml = RuleServiceProxy.ExecuteXmlRule("RuleSetCode", UserIdentity.TenantID, inputXml);
```

Here the `inputXml` will contain the necessary variable values to successfully execute the rule. The output will also be an Xml containing the modified values.



E.g:

```
<root>
  <amount type="Decimal">150</amount>
  <type type="String">Cash</type>
</root>
```

Business Rule APIs

You can also use the below APIs to create rule and map activities to business rules.

```
interface IRuleSetMetadataService
{
    /// <summary>
    /// Adds the rule activity.
    /// </summary>
    /// <param name="activityDefinition">The activity definition.</param>
    string AddRuleActivity(RuleActivityDefinition activityDefinition);

    /// <summary>
    /// Creates the rule set metadata.
    /// </summary>
    /// <param name="ruleSetMetadata">The rule set metadata.</param>
    /// <returns></returns>
    string CreateRuleSetMetadata(RuleSetMetadata ruleSetMetadata);

    /// <summary>
    /// Deletes the rule activity.
    /// </summary>
    /// <param name="activityId">The activity id.</param>
    /// <param name="rulesetCode">The ruleset code.</param>
    /// <param name="tenantId">The tenant id.</param>
    void DeleteRuleActivity(string activityId, string rulesetCode, string tenantId);

    /// <summary>
    /// Deletes the rule set metadata.
    /// </summary>
    /// <param name="rulesetCode">The ruleset code.</param>
    /// <param name="tenantId">The tenant id.</param>
    void DeleteRuleSetMetadata(string rulesetCode, string tenantId);

    /// <summary>
    /// Gets all rule set metadata.
    /// </summary>
    /// <param name="tenantId">The tenant id.</param>
    /// <returns></returns>
    Dictionary<string, RuleSetMetadata> GetAllRuleSetMetadata(string tenantId);

    /// <summary>
    /// Gets the rule activity definitions.
    /// </summary>
    /// <param name="rulesetCode">The ruleset code.</param>
    /// <param name="tenantId">The tenant id.</param>
    /// <returns></returns>
    Dictionary<string, RuleActivityDefinition> GetRuleActivityDefinitions(string rulesetCode, string
tenantId);

    /// <summary>
```



```
/// Gets the rule set metadata.  
/// </summary>  
/// <param name="rulesetCode">The ruleset code.</param>  
/// <param name="tenantId">The tenant id.</param>  
/// <returns></returns>  
RuleSetMetadata GetRuleSetMetadata(string rulesetCode, string tenantId);  
  
/// <summary>  
/// Updates the rule set metadata.  
/// </summary>  
/// <param name="ruleSetMetadata">The rule set metadata.</param>  
void UpdateRuleSetMetadata(RuleSetMetadata ruleSetMetadata);  
  
/// <summary>  
/// Returns true or false for the given entities has the specified rule configured.  
/// </summary>  
/// <param name="entityIds">array of entity ids</param>  
/// <param name="ruleType">rule type</param>  
/// <returns></returns>  
Dictionary<string, bool> CheckEntityHasRule(string[] entityIds, EntityRuleType ruleType);  
}  
  
interface IRuleService  
{  
  
    /// <summary>  
    /// Executes the given rule with the specified inputXml and returns the outputXml  
    /// </summary>  
    /// <param name="ruleSetCode">ruleSetCode to execute</param>  
    /// <param name="tenantId">tenant Id</param>  
    /// <param name="inputXml">input rule variable/value xml</param>  
    /// <returns></returns>  
    string ExecuteXmlRule(string ruleSetCode, string tenantId, string inputXml);  
  
    /// <summary>  
    /// Get the rule variables as Xml  
    /// </summary>  
    /// <param name="ruleSetCode"></param>  
    /// <param name="tenantId"></param>  
    /// <returns></returns>  
    string GetRuleVariablesXml(string ruleSetCode, string tenantId);  
  
    /// <summary>  
    /// Copies the ruleset from parent tenant.  
    /// </summary>  
    /// <param name="ruleSetCode">The rule set code.</param>  
    /// <param name="tenantId">The tenant id.</param>  
    /// <param name="userId">The user id.</param>  
    /// <returns></returns>  
    bool CopyRulesetFromParent(string ruleSetCode, string tenantId, string userId);  
}
```

Tips for crafting a business rule:

- Analyze the scenario and come up with input & output variables
- Create a Activity class to register/populate variables
- Create a rule class for interacting with the rule engine/input/output parameters
- Configure the rule using Silverlight UI.
- Execute the rule at the required place.

Business Rule Auditing

CelloSaaS Business Rule Engine offers two types of auditing/debugging experience to the developers.

In Memory audit log
File based audit log

During the development time, the developers can use In-Memory audits as this will be easier to trace and diagnose the rules execution problems (URL: /Audits/BusinessRules). In production you can switch over to file based audit logging by using the below configuration.

Open the Unity.config file and find the below highlighted line.

```
<container name="DataAccess">
  <types>
    <type type="ILogSink" mapTo="MemoryLog" name="ILogSink"/>
```

Change **MemoryLog** to **ELLog** for file based logging using Enterprise Library Application Logging Block which can be configured in the web.config (should contain a category named "RulesLog").

```
<loggingConfiguration name="Logging Application Block" tracingEnabled="true"
  defaultCategory="General" logWarningsWhenNoCategoriesMatch="true">
  <categorySources>
    <add switchValue="All" name="RulesLog">
      <listeners>
        <add name="Rules Listner" />
      </listeners>
    </add>
  </categorySources>
</loggingConfiguration>
```

Configuring WCF Rule Service for Silverlight UI:

Silverlight UI requires a web service to fetch/save the business rules. CelloSaaS package will already have valid configuration for development environment in localhost. However in production or when hosting the CelloSaaS WCF you need to change the URL pointing to the WCF services in the appSettings section of the Web.config as below.

```
<add key="BusinessRuleWcfURL" value="http://cellosaas.wcf.url/BusinessRuleManagement"/>
```

How to Guide: [Entity Based Business Rules](#)

How to Guide: [Open Business Rules](#)

How to Guide: [XML Rule](#)

Reporting Service

CelloSaaS Reporting Services provides a compact ready-to-use tool and services to help you create, manage reports by tenants and sub tenants with the entities/Objects used in the application. It is an AD-Hoc Query based Reports builder, with web based easy to use interface and SQL alike features which helps the business Experts and non IT professionals to easily create and manipulate records.

Reporting Services is a Client-Server based reporting platform that provides comprehensive reporting functionality. Reporting Services includes a complete set of tools for you to create, manage, and deliver reports, and APIs that enable developers to integrate or extend data and report processing in multi tenant SaaS environment. The Query Builder tool brings in the simplified query building experiencing without losing the SQL server functionalities such as Relationships, Query forming Clauses such Select, Where, Conditions, Group By etc. The important aspect of the CelloSaaS Reporting Services is that, it injects multi tenancy, Data Scoping, Data Security and mostly importantly getting Access to the extended fields with any type of implementations Refer [Extended Fields].

With Reporting Services, you can create interactive, tabular, graphical from relational data sources. Reports can include rich data visualization, including charts, maps. You can publish reports, schedule report processing, or access reports on-demand. You can select from a variety of viewing formats, export reports to other applications such as Microsoft Excel and PDF. The reports that you create can be saved separately and viewed later or provide as a data source for Charting or Report Builder. The data Scope will be automatically applied while executing the saved/Running queries based on the logged in User Context.

Example

☞ Step 1: Select * from Payroll

This Query will return a different result set for a Role called manager, where as it will behave differently for “Employee Role”, Simple because both the roles has different data scope assigned.

☞ Step 2: See Chapter:DataScope for more information.

Note: This is not intended to use as Standalone Reporting Services tool for enhanced usage.



Figure 16-1 – Reporting Service Advantages

Typically, the primary tenants builds the complex queries or the required queries and saves it. The users of the tenant can use this query to execute and produce the reports. Child tenants or sub tenants can also copy these queries created by their respective parents and use it as it is or they can change the queries as per their requirements.

Limitations

Data can be Queried only on the registered entities[Tables] in the System.

No Complex queries can be built using CelloSaaS Reporting Services

SQL Objects such as multiple Joins, Sub Queries, SQL Functions etc are not possible

CelloSaaS Reporting Services has the following Components



QueryBuilder

This supports building the ad-hoc queries based on the entities.

ChartBuilder

This supports building charts based on a variety of sources like Text / StoredProcedure or consuming the QueryBuilder as a source

ReportBuilder

This supports creating ad-hoc reports which can contain various report objects. In this version (4.0), only chart and table source report objects are supported.

17 QUERY BUILDER

Query builder library provides the following functionalities:

Query any entity

- Map, reference and audit fields via configuration file for better control of data generated as a result of query execution
- Create Queries with extension fields in the select fields list
- Create Queries with pickup list fields in the select and where clause fields
- Export Query result in 3 formats, viz. CSV, Excel and PDF
- Field Level privilege and datascope for fine grained control of query results.
- Automated tenant isolation for query results.

Pre-Requisites

QueryBuilder Configuration

The QueryBuilder Configuration file [QueryBuilderConfiguration.config] should have the entire foreign key references [Foreign key references] for each entity registered.

Entities

The “Entity” table has an ‘Entity_SchemaTableNameConnectionString’ field which has to specify the connection string name associated with the particular entity. If no value is entered, the default value of ‘CelloSaaSConnectionString’ is used.

Entity Fields

The“EntityFields” table has a field ‘MappedColumnName’ which has the name of the physical column name for the particular entity field. This will be used for the application of the field level privilege for the particular field while query execution.

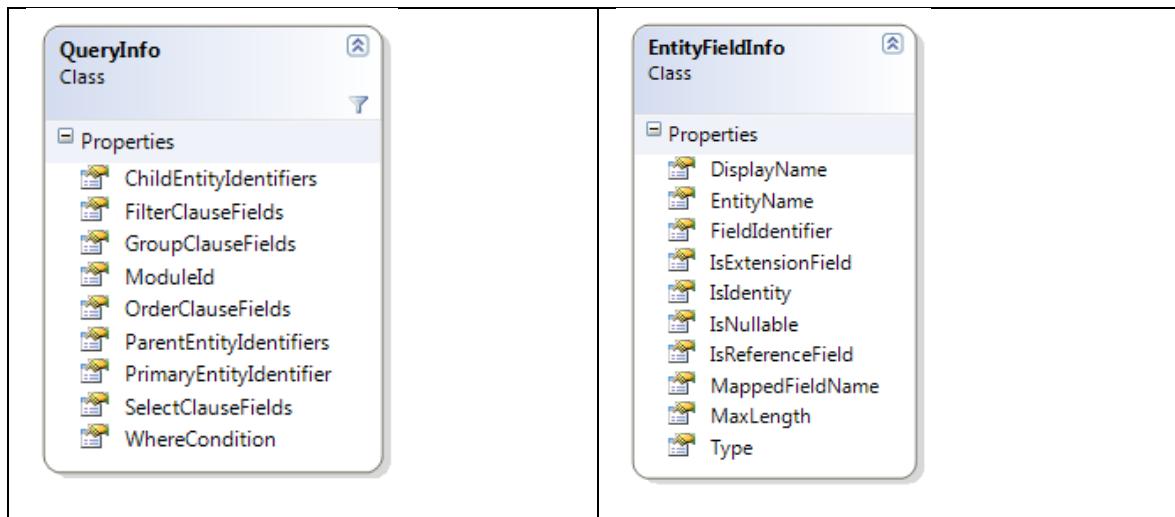
DataScope

For the datascope queries, use the entity id as the alias for the corresponding schematblenames in the datascope queries.

Others

All the Entities registered in the Entity table should have the Tenant ID or Tenant Code as a field in the corresponding schema table. Only the fields which have the registered field identifier in the entity fields will be applicable for the field privilege application in the resultant query

The data to be passed to consume the services should confirm to the following object models.
The model namespace is: CelloSaaS.QueryBuilderLibrary.Model



Sample Entity and Entity Fields Mapping

For the Entity “Address”, the corresponding entries in the Entity and EntityFields tables are

Table : Entity	
Field	Value
Entity_ID	Address
Entity_ExtendedSchemaTableName	CelloSaas.Address
Entity_PrimaryKeyName	Address_ID
Entity_SchemaTableConnectionStringName	CelloSaaSConnectionString (OR) NULL

Table : EntityFields	
Field	Value
EntityFields_EntityId	Address
EntityFields_Id	Address_Address1
EntityFields_Name	Address1
EntityFields_MappedColumnName	Address_Address1

QueryBuilderConfiguration.config file

```

<QueryBuilderConfiguration>
<AuditFieldConfiguration>
<AuditFields>
<Field ..>
</Field>
</AuditFields>
</AuditFieldConfiguration>
<EntityFieldConfiguration>
<Entity ..>

```

```
<Field ..>
</Field>
</Entity>
...
</EntityFieldConfiguration>
</QueryBuilderConfiguration>
```

Entity Field element Attributes

Identifier = Entity Field Identifier from EntityFields Table
 ReferenceColumn = Primary Key column for this mapped reference column
 DisplayColumn = The column from which the value to be displayed is to be taken from
 ReferenceTable = Reference Table Name [Primary key table]
 Type = Either of {Table , PickupList}
 MappedColumnName = The database column Name
 ReferenceEntity = The Entity Identifier for the reference table name
 DisplayName = The display name for this field
 ReferencePickListName = Name of the pickuplist
 ReferencePickListId = Pickup List Identifier

Entity element attributes

Identifier= Entity Identifier for the entity
 TableName = SchemaTableName for this entity
 Schema= Schema in which the Schema Table belongs to
 StatusColumn = The status column name
 TenantColumn = The tenant identifier or tenant code column name

Sample Entity & EntityField Configuration Elements

```
<Entity Identifier="EmployeeNumberConfiguration" Schema="Silica"
TableName="EmployeeNumberConfiguration" StatusColumn="Status" TenantColumn="TenantID">
<Field Identifier="" MappedColumnName="CompanyID" Type="Table"
ReferenceTable="OrganizationHierarchy" DisplayColumn="OrganizationHierarchyName"
ReferenceColumn="OrganizationHierarchyID" DisplayName="Organization Hierarchy Name"
ReferenceEntity="HierarchyNode"/>
</Entity>
```



The QueryBuilderConfiguration.config file is case sensitive and values entered for attributes should be of the proper casing with that of the real names.

There should not be any comments in the form of [<!-- ... -->] in the configuration file.

Web.config Settings

The following are the web.config settings related to the QueryBuilder Library. Date Time format in appsettings to be set so that the output date time formatting will be done.

```
<add key="ApplicationDirectoryPath" value="C:\\...\\CelloSaaSApplication"/>
```

The QueryBuilderConfiguration.config file to be added to the Config folder in the application and its path to be given configuration file.

```
<add key="DateFormat" value="mm/dd/yyyy"/>
```

DataSources

The DataSource is used as a source provider for the Chart and Report Builder. DataSources can be any of the following types.

Text Query

Stored Procedure

Built-In Query sources [Queries built in using the CelloSaaS Query Builder].

When providing the SourceContent for any datasource, the SourceContentViewModel needs to be used for setting the contents for the SourceContent.

```
public class SourceContentViewModel
{
    public string DataSource_Id { get; set; }
    public string DataSource_Name { get; set; }
    public string SourceContent_Id { get; set; }
    public string SourceContent { get; set; }
    public string SourceContentTypes { get; set; }
    public string BuiltInQuerySources { get; set; }
    public List<Parameters> parameters { get; set; }
    public SourceContentViewModel()
    {
        parameters = newList<Parameters>();
    }
}
public class Parameters
{
    public string ParamName { get; set; }
    public string ParamValue { get; set; }
}
```

Limitations

Query builder doesn't have the capability to fetch records from database from disparate server

17.1 Chart builder

Anatomy of a Chart

The chart is sectioned as per the following figure.

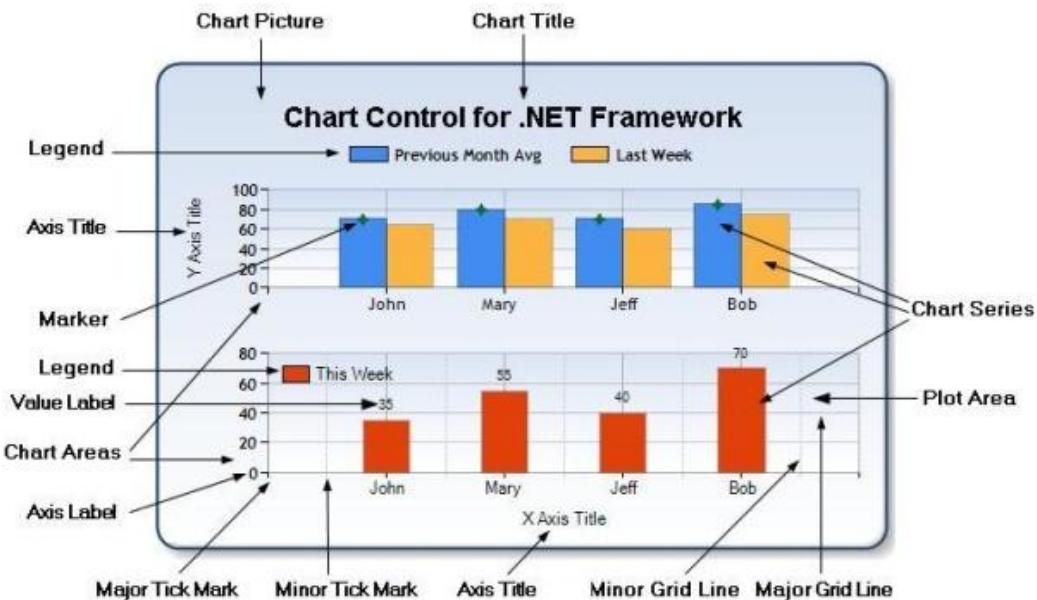


Figure 17-1 – Anatomy of a Chart

Each section will have its own properties. The properties are therefore mapped to sections via the `ChartSectionProperties`.

To add new properties for a chart or a common property for a chart, the following steps are to be followed,

☞ Step 1: Add a new section to the chart if needed

CharSections_Id	CharSections_SectionName	CharSections_AddedBy	CharSections_AddedOn	CharSections_UpdatedBy	CharSections_UpdatedOn	CharSections_Status
1	Title	B590CD25-3093-DF11-8DEB-001EC9DAB123	2011-11-15 12:06:57.437	NULL	NULL	1
2	Legend	B590CD25-3093-DF11-8DEB-001EC9DAB123	2011-11-15 12:07:06.650	NULL	NULL	1
3	Chart	B590CD25-3093-DF11-8DEB-001EC9DAB123	2011-11-15 12:07:14.183	NULL	NULL	1

☞ Step 2: Add the Property in the ChartProperty Table,

ChartProperty_Id	ChartProperty_PropertyName	ChartProperty_AddedBy	ChartProperty_AddedOn	ChartProperty_UpdatedBy	ChartProperty_UpdatedOn	ChartProperty_Status
45C2E388-24CB-4980-9F49-05331541C11B	PieDrawingStyle	B590CD25-3093-DF11-8DEB-001EC9DAB123	2012-01-31 15:50:00.550	NULL	NULL	1

☞ Step 3: Now, map the chart property to the chart section via the `ChartSectionProperties`

ChartSectionProperties_Id	ChartSectionProperties_PropertyId	ChartSectionProperties_SectionId	ChartSectionProperties_TenantId	ChartSectionProperties_AddedBy	ChartSectionProperties_Pro
1	73F05809-6B4C-42CB-A893-2C15205EBE67	45C2E388-24CB-4980-9F49-05331541C11B	5EFD8340-540F-E111-B266-B8AC6F2D3722	NULL	B590CD25-3093-DF11-8DEB-001EC9DAB123 2012-01-31 17:14:48.173

☞ Step 4: Then the partial views to be updated with these properties and then consumed in the controller.



Any chart section property without a tenant identifier is globally available for all the tenants.

The following chart types are supported in the CelloSaaS Chart Builder:

- Point
- Line
- Bar
- Column
- Pie

Each chart can have a partial view that renders the properties for that chart type.

```
public class ChartDetails
{
    public string AddedBy;
    public DateTime AddedOn;
    public string ChartId;
    public string ChartName;
    public Dictionary<string, ChartAxisFields> ChartsAxisFields;
    public string ChartSourceId;
    public Dictionary<string, ChartPropertyValues> ChartsProperties;
    public string ChartType;
    public Dictionary<string, Dictionary<string, ChartPropertyValues>> OtherProperties;
    public bool Status;
    public string TenantId;
    public string UpdatedBy;
    public DateTime UpdatedOn;
    public ChartDetails();
}
```

Charts Listing

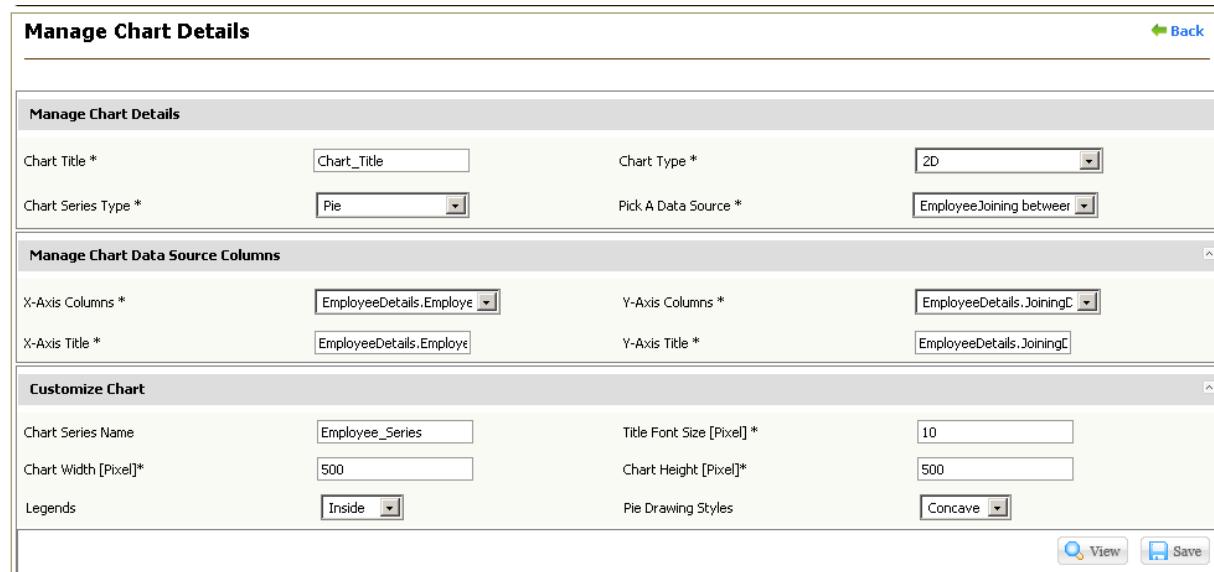


The screenshot shows a web-based application titled "Chart Builder". At the top right is a blue "Add" button with a plus sign. Below it is a dark green header bar with the text "Manage Charts". Underneath is a white content area. On the left, there's a sidebar labeled "Available Charts" containing a dropdown menu set to "Employee Status Chart". To the right of the sidebar are several action buttons: "Export" (red), "Edit" (blue), "View" (green), and "Delete" (orange). The main content area is currently empty, indicating no charts have been created yet.

Figure 17-2 – Chart Builder

The generated Chart can be exported in both Image and PDF formats. The image format supported in this version is png only.

Chart Creation



The screenshot shows a detailed configuration page for "Manage Chart Details". It has a "Back" button at the top right. The main area is divided into sections: "Manage Chart Details", "Manage Chart Data Source Columns", and "Customize Chart". In the "Manage Chart Details" section, fields include "Chart Title *" (Chart_Title), "Chart Type *" (2D), "Chart Series Type *" (Pie), and "Pick A Data Source *" (EmployeeJoining between). In the "Manage Chart Data Source Columns" section, "X-Axis Columns *" is set to "EmployeeDetails.Employee" and "Y-Axis Columns *" is set to "EmployeeDetails.JoiningDate". In the "Customize Chart" section, fields include "Chart Series Name" (Employee_Series), "Title Font Size [Pixel] *" (10), "Chart Width [Pixel]*" (500), "Y-Axis Title *" (EmployeeDetails.JoiningDate), "Chart Height [Pixel]*" (500), "Legends" (Inside), "Pie Drawing Styles" (Concave), and "Save" and "View" buttons at the bottom right.

Figure 17-3 – Manage Chart Details

Apart from the chart's specific properties, the charts also have the following properties in common.

As shown in the figure above, the chart's axis title can also be customized by providing a different name than that of the data source's name.

DataBinding The data for the chart is obtained as a datatable and then bound to the chart.

TableSources Builder

The functionality of this Tablesource Builder is to generate the Tablesources to be consumed by the Report objects. The Table source builder is used to map the datasources to the tablesources. Once a table source is mapped to a datasource, the datasource for that table source cannot be changed. Any number of table sources can be mapped to a datasource.

Table Sources Listing



Welcome, euroset ! [Log Off] | Settings | Help | Support

Manage Table Sources

Manage Table Sources

Available Table Sources	Employee Status Table

Edit Delete

Figure 17-4 – Manage Table Source

Table Source Detail View



Welcome, euroset ! [Log Off] | Settings | Help | Support

ManageTableSources

Manage Table Source

Table Source Name	Employee Status Table
Table Source Description	
Query Source	EmployeeStatusCount

Back Save

Figure 17-5 – Manage Table Source

17.2 Dynamic variables in query builder

This is a new feature in QueryBuilder that enables the queries to be built in with parameters analogous to that of parameters in SQL Queries. This is used only for the formation of filter clauses in the Queries. When a filter clause in the query is built with a dynamic variable, upon execution the user will be prompted for a value for this variable. The input value is passed on to the filter condition of the query.

Advantages

This feature enables the queries to be executed based on user inputs than going back and editing the query filter conditions for varying the results, thereby bringing in the dynamism in the query results. This feature will be available in V3.4 release to the report builder so that reports can be made dynamic.

How it has to be called/used by the developer?

This feature is not consumable by developers. This is for the end users of the Query Builder Library

How it could be customized/enhanced by the developer?
Sealed Feature

Limitations of the feature:
Nil

17.3 Report builder

The Sources for the ReportBuilder can be any one of the following:

- Table Source [Obtained from a DataSource]
- Chart Source [Obtained from ChartBuilder Charts]
- Text Source [Plain Text] or an Image Source*

* Not supported in Version 3.2

Model

```
public class Report : BaseEntity
{
    public string CreatedBy;
    public DateTime CreatedOn;
    public string Description;
    public string Name;
    public string ReportId;
    public Dictionary<string, ReportObject> ReportObjects;
    public bool Status;
    public string TenantId;
    public string UpdatedBy;
    public DateTime UpdatedOn;
    public Report();
}

public class ReportObject
{
    public string CreatedBy;
    public DateTime CreatedOn;
    public ReportObjectType ObjectType;
    public int Ordinal;
    public Dictionary<string, TableResultColumns> ReportColumns;
    public string ReportId;
    public string ReportObjectId;
    public string SourceId;
    public IObjectSource SourceObject;
    public bool Status;
    public string TenantId;
    public string TypeId;
    public string UpdatedBy;
    public DateTime UpdatedOn;
    public ReportObject();
}
```

Reports Listing View

Figure 17-6 – Report Builder

Report Builder has the following components:

Report Designer

This is a Designer canvas for the addition of the report objects to the report and also to configure or customize the report object. This version of the Report Builder currently supports only the Table and the Chart Report Objects.

The Report Builder has a drag and drop UI. The Report Objects are dragged onto the designer canvas and the report objects are persisted. The report object sources are bound after the addition of the report object to the designer and then the other properties are set and persisted on a per-report object basis. The dropped report objects can be re-ordered and then their positions will be persisted in the database.

Each of the report objects will contain a section header and if the source forms a table source for that particular report object, we can set the number of records that have to be displayed during the executed view of the report.

Figure 17-7 – Report Designer

Report Viewer

This is a viewer component where the reports are executed and then the executed results are displayed here. The table report objects are paged based on the properties configured for that report object.

The screenshot shows the CelloSaaS Report Preview interface. At the top, there is a navigation bar with links: Manage Users, Manage Access, Manage Configurations, Audits, Reporting, Welcome, Log Off, Settings, Help, and Support. Below the navigation bar, there is a "Report Preview" section titled "Employee Status Count Report". This section contains a table titled "Department Employees table" and a chart titled "Total Employees Chart".

Department Employees table

Track.Name	COUNT(Designation.DesignationName)
Development	14
Management	8
System	2
Testing	19

Show 10 entries Showing 1 to 4 of 4 entries

Total Employees Chart

Design_Track

A bar chart titled "Design_Track" showing the count of employees by designation. The Y-axis is labeled "COUNT(EmployeeDetails.EmployeeNumber)" and ranges from 0 to 15. The X-axis is labeled "Designation.DesignationName". The chart has a legend indicating "Track" (green bars). The data points are:

Designation.DesignationName	Count
Account Manager	2
Assistant Manager	5
CEO	1
Delivery Manager	8
Manager	1
Manager	2
Manager	2
Onsite Project Manager	1
President	1
Project Leader	12
Project Manager	7
System Administrator	2
Technical Architect	2

Designation.DesignationName

Figure 17-8 – Report Preview

The report view supports exporting the report in a PDF format. The components that make up the report are all exported in a PDF file.

Report Print View:

The report can be viewed without any pagination for the table sources. This view is provided to aid in printing the reporting directly.

Employee Status Count Report

[Back](#)

Department Employees table

Track.Name	COUNT(Designation.DesignationName)
Development	14
Management	8
System	2
Testing	19

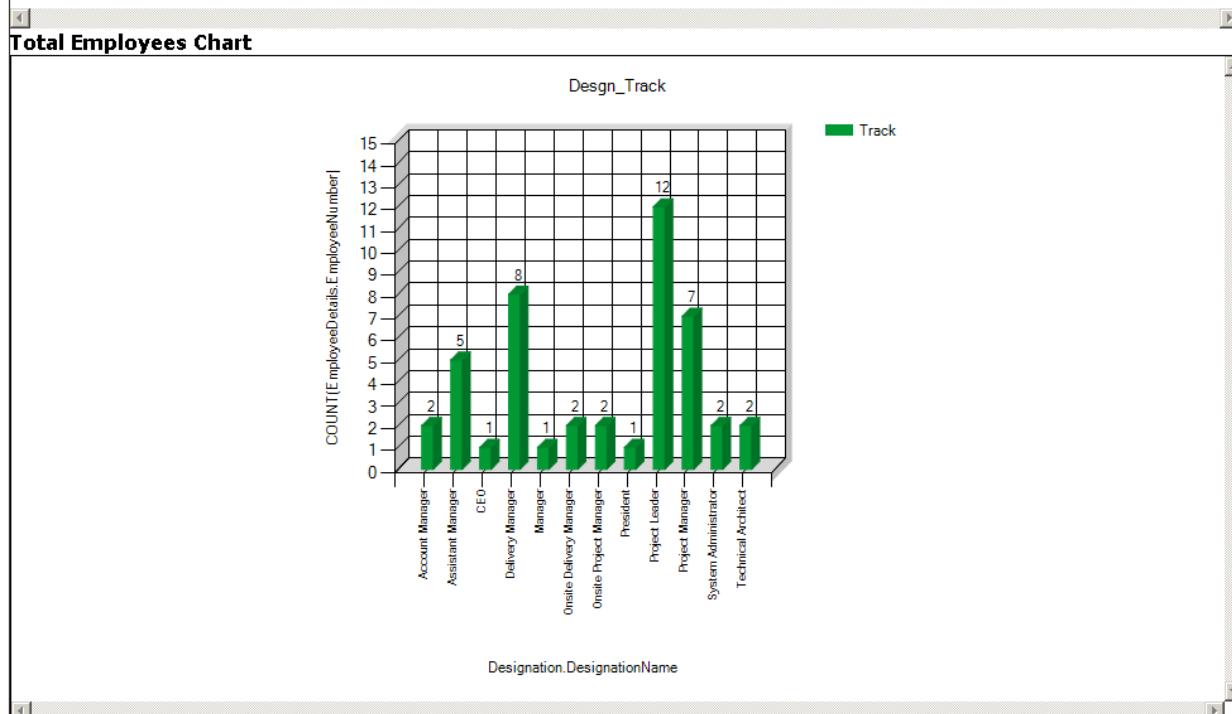


Figure 17-9 – Report Preview

Export PDF Report Sample

Employee Status Count Report

Department Employees table

Track.Name	COUNT(Designation.DesignationName)
Development	14
Management	8
System	2
Testing	19

Total Employees Chart

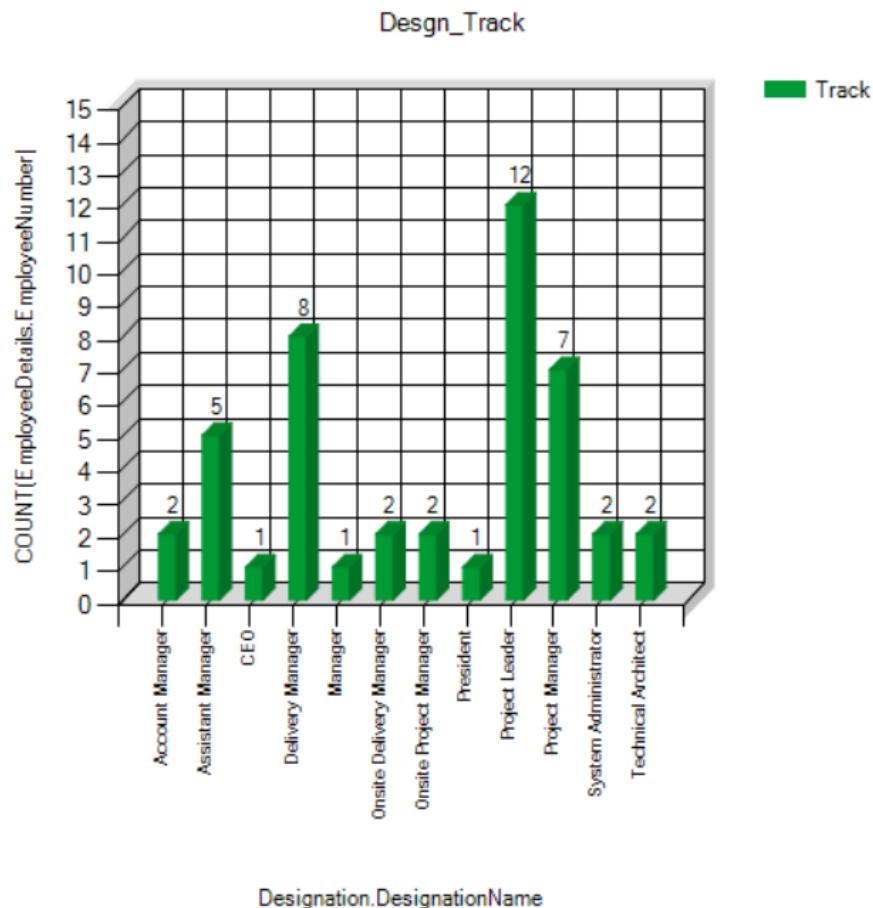


Figure 17-10 – Report Preview

Table Report Objects

These are tabular data based report objects. The following are the customizations possible in this type of report object:

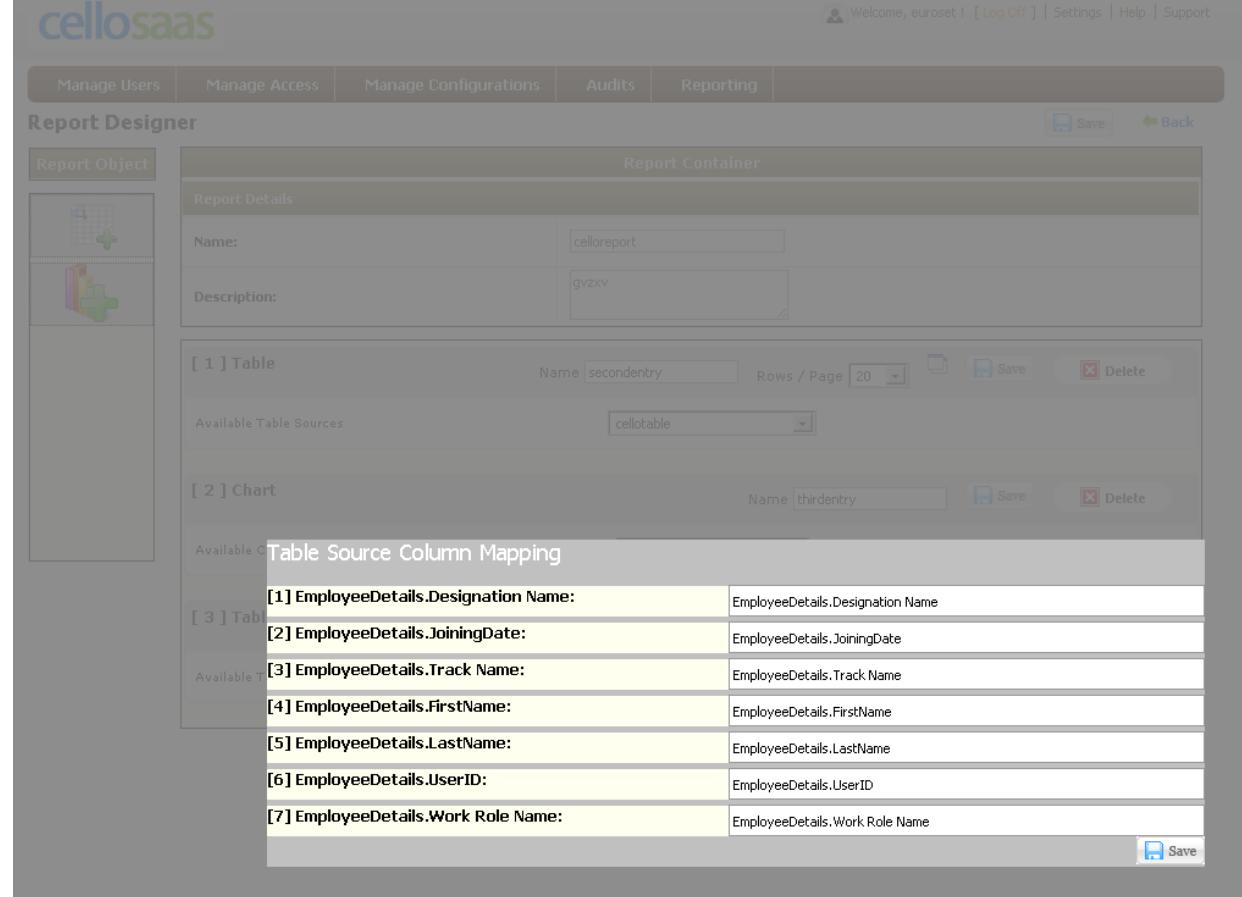
The Columns for the Table Source Report Object can be configured via the change of display names and re-ordering of the columns can be made from the report designer itself per report object basis. The number of records that are to be displayed when the report is executed can also be configured.

Model

```
public class TableSource : IObjectSource
{
    public string CreatedBy;
    public DateTime CreatedOn;
    public string DataSourceld;
    public DataSource DataSourceValues;
    public string Description;
    public string Id;
    public string Identifier;
    public string Name;
    public Dictionary<string, TableResultColumns> ReportColumns;
    public bool Status;
```

```
public string TenantId;
public string UpdatedBy;
public DateTime UpdatedOn;
public TableSource();
}
```

Configuring the Table Source from the Report Object is shown in the figure below,



The screenshot shows the 'Report Designer' section of the CelloSaaS platform. On the left, there's a sidebar with icons for 'Manage Users', 'Manage Access', 'Manage Configurations', 'Audits', and 'Reporting'. The 'Reporting' tab is selected, showing a 'Report Container' with a 'Report Details' panel containing fields for 'Name' (celloreport) and 'Description' (gyczxy). Below it is a 'Table' section with a 'Name' field set to 'secondentry', a 'Rows / Page' dropdown set to 20, and buttons for 'Save' and 'Delete'. An 'Available Table Sources' dropdown is set to 'cellotable'. There's also a 'Chart' section with a 'Name' field set to 'thirdentry' and similar save/delete buttons. The main area is titled 'Table Source Column Mapping' and contains a table mapping columns from 'EmployeeDetails' to 'EmployeeDetails'. The table has 7 rows:

[1] EmployeeDetails.Designation	Name:	EmployeeDetails.Designation Name
[2] EmployeeDetails.JoiningDate:		EmployeeDetails.JoiningDate
[3] EmployeeDetails.Track	Name:	EmployeeDetails.Track Name
[4] EmployeeDetails.FirstName:		EmployeeDetails.FirstName
[5] EmployeeDetails.LastName:		EmployeeDetails.LastName
[6] EmployeeDetails.UserID:		EmployeeDetails.UserID
[7] EmployeeDetails.WorkRole	Name:	EmployeeDetails.Work.Role Name

At the bottom right of the mapping table is a 'Save' button.

Figure 17-11 – Table Source Column Mapping

Chart Report Object

Only the source chart for this report object can be changed with respect to this report object

This ReportBuilder can be extended for any datasource as long as that Source implements the IObjectSource interface and implements its properties.

```
public interface IObjectSource
{
    string CreatedBy { get; set; }
    DateTime CreatedOn { get; set; }
    string Id { get; set; }
    string Name { get; set; }
    bool Status { get; set; }
    string TenantId { get; set; }
    string UpdatedBy { get; set; }
    DateTime UpdatedOn { get; set; }
}
```



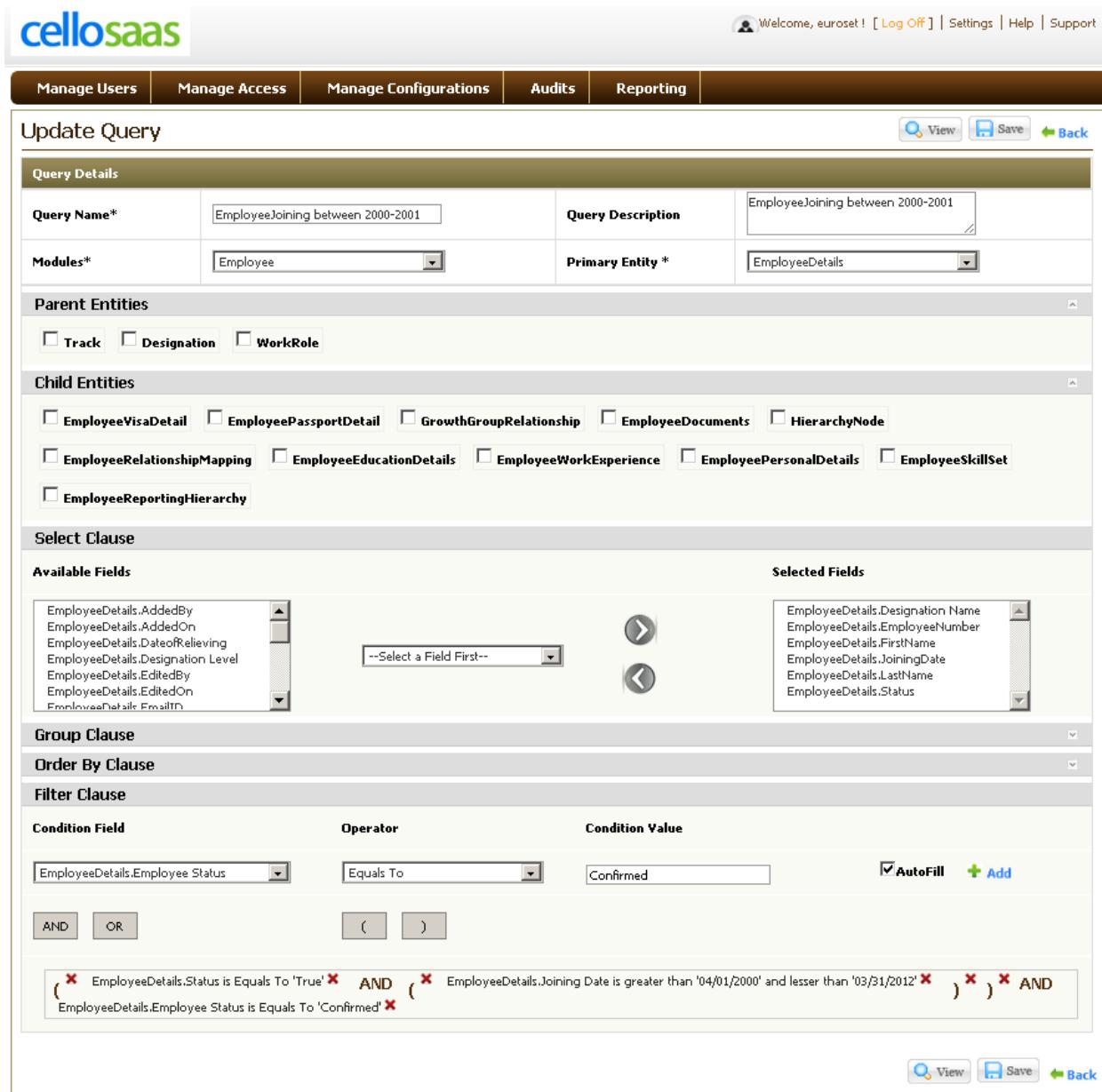
The DataSources is also extensible as long as that datasource's services and DAL inherits from the `IDataSource` Interface.

```
public interface IDataSource
{
    string AddSource(IObjectSource newSource);
    bool CheckDuplicateSourceName(string sourceName, string tenantId, string existingSourceId);
    int DeleteSource(string sourceId, string tenantId);
    List<string> FindSourceNames(string partialName, string tenantId);
    Dictionary<string, IObjectSource> GetAllSources(string tenantId);
    Dictionary<string, string> GetShallowSources(string tenantId);
    IObjectSource GetSource(string sourceId, string tenantId);
    Dictionary<string, IObjectSource> GetSources(List<string> sourceIds, string tenantId);
    Dictionary<string, IObjectSource> GetSourcesByContentId(string contentId, string tenantId); int
    UpdateSource(IObjectSource updatedSource);
}
```

There is a `ReportBuilder.config` file that has the mapping of the Report Object source and its related services, which will be automatically resolved during the runtime.

```
<Report>
    <TypeSourceMap>
        <type name="Table" assemblyName="CelloSaaS.Reporting.DataSources"
typeName="CelloSaaS.Reporting.DataSources.Services.TableSourceService" />
        <type name="Text" assemblyName="CelloSaaS.Reporting.DataSources"
typeName="CelloSaaS.Reporting.DataSources.Services.TextSourceService" />
        <type name="Image" assemblyName="CelloSaaS.Reporting.DataSources"
typeName="CelloSaaS.Reporting.DataSources.Services.ImageSourceService" />
    </TypeSourceMap>
</Report>
```

In QueryBuilder, the support for the parenthesis is implemented from Version 3.2 onwards. This enables nested parenthesis in the filter conditions and also enables better control of the filter conditions that govern the results produced by the querybuilder.



The screenshot displays the 'Update Query' page of the CelloSaaS QueryBuilder. The top navigation bar includes links for Manage Users, Manage Access, Manage Configurations, Audits, and Reporting. The main content area is titled 'Update Query' with buttons for View, Save, and Back.

Query Details:

- Query Name:** EmployeeJoining between 2000-2001
- Query Description:** EmployeeJoining between 2000-2001
- Modules:** Employee
- Primary Entity:** EmployeeDetails

Parent Entities: Track, Designation, WorkRole

Child Entities: EmployeeVisaDetail, EmployeePassportDetail, GrowthGroupRelationship, EmployeeDocuments, HierarchyNode, EmployeeRelationshipMapping, EmployeeEducationDetails, EmployeeWorkExperience, EmployeePersonalDetails, EmployeeSkillSet, EmployeeReportingHierarchy

Select Clause:

Available Fields: EmployeeDetails.AddedBy, EmployeeDetails.AddedOn, EmployeeDetails.DateofRelieving, EmployeeDetails.Designation Level, EmployeeDetails.EditedBy, EmployeeDetails.EditedOn, EmployeeDetails.EmailID

Selected Fields: EmployeeDetails.Designation Name, EmployeeDetails.EmployeeNumber, EmployeeDetails.FirstName, EmployeeDetails.JoiningDate, EmployeeDetails.LastName, EmployeeDetails.Status

Group Clause:

Order By Clause:

Filter Clause:

Condition Field: EmployeeDetails.Employee Status
Operator: Equals To
Condition Value: Confirmed
 AutoFill + Add

Logical Operators: AND, OR, (,)

Filter Condition: (EmployeeDetails.Status is Equals To 'True' AND EmployeeDetails.Joining Date is greater than '04/01/2000' and lesser than '03/31/2012'))) AND EmployeeDetails.Employee Status is Equals To 'Confirmed'

Figure 17-12 – Update Query

17.4 Product Analytics

CelloSaaS analytics helps to capture various information about the application and its usage, users, time duration, visitors, Components, features, modules etc. This can help the managers and administrators of the product to analyse and understand the usage patterns of the application by various tenants and its users and optimize the product accordingly.

This information is captured on the runtime by the framework by simply enabling the Product Analytics settings; similarly it could be turned off when it is not required.



CelloSaaS analytics is not just a tool for measuring web traffic and usage but can be used as a tool for business and market research, and to assess and improve the effectiveness of the product. It helps the ISVs to understand how many users accessing the application/Module in a given point of time, or how well the new features are used by the tenants and their users and so on.

This feature is available in namespace: CelloSaaS.View.[TrackUsage](#)

The [TrackUsage / DisableTrackUsage](#) attribute is to be used on the controller or on the action on which the requests to be tracked or not tracked.

Tracking

A. TrackUsage Attribute in Controller:

```
[CelloSaaS.View.TrackUsage(TraceLevel = CelloSaaS.View.TraceLevels.Verbose)]  
public class RolesController : CelloSaaS.View.CelloController  
{ ... }
```

B. TrackUsage Attribute in Action Method:

```
[CelloSaaS.View.TrackUsage(TraceLevel = CelloSaaS.View.TraceLevels.Verbose)]  
public ActionResult UserList()  
{ ... }
```

C. To Disable Tracking:

```
[DisableTrackUsage]  
public ActionResult UserList()  
{ ... }
```

The attribute when used in the controller, causes all the requests for all the actions in that controller to be logged in the database. The attribute when used in the action method, causes all the requests for this particular action method to be logged. The [DisableTrackUsage](#) attribute is used to avoid the tracking of the requests for the particular action method or for all the methods in the controller.

ConnectionStrings

The logging of the usage can be stored in a separate database [similar to that of the UserConnectionString], with the use of the ProductAnalyticsConnectionString in the web.config file in the <connectionStrings> section.

Sample ConnectionString for SQL Server Database based analytics logging:
<add name="ProductAnalyticsConnectionString" connectionString="Data Source=Server;Initial Catalog=Database;User Id=UserId;Password=Password;" providerName="System.Data.SqlClient"/>
Sample ConnectionString for MySQL Server Database based analytics logging
<add name="ProductAnalyticsConnectionString" connectionString="Data Source=Server;Initial Catalog=Database;User Id=UserId;Password=Password;" providerName=" MySql.Data.MySqlClient"/>

Enable / Disable Product Analytics

To enable and disable the product analytics, we have two options,

Application Based

To enable or disable the product analytics on a application basis, the appsettings should be set in the web.config as follows, `<add key="EnableProductAnalytics" value="true" />`

Tenant Based

To enable or disable on a per tenant basis, setting to be made from the tenant settings

Tenant Settings	
Save	
Tenant Settings	
Theme	<input type="text" value="CelloSkin"/>
Logo	 <input type="button" value="Choose File"/> No file chosen
Product Analytics:	<input checked="" type="checkbox"/>
Password Expiration Days	<input type="text" value="0"/>
Home Realm:	<input type="text" value="anand@tps.com"/>
WCF Secret Key	<input type="text" value="*****"/>
Save	

Figure 17-13 – Tenant Settings

Unchecking in the Product Analytics Checkbox does not log the analytics data to the database and vice versa.

Trace Levels

None: No analytics information is logged at this level

UrlTrace: The following data are available in this level of tracing:

- ActionName
- ControllerName
- TimeStamp
- CurrentUrl
- UserId
- TenantId

Verbose: The following data are available in this level of tracing:

- ActionName
- UserId
- ControllerName
- TimeStamp
- CurrentUrl
- TenantId
- ResponseStatusCode
- ResponseDescription
- ExceptionDetails
- MachineName
- RequestMethod
- RequestMode
- IPAddress

The following additional data are captured and logged

BrowserDetails [browser type , Platform, EcmaScriptVersion]

RequestMethod can be either AJAX or WEB. The captured data are stored in the [ProductAnalytics](#) table in the database.

Viewing Analytics Data

The product analytics log data can be accessed via the following API:

```
CelloSaaS.ServiceProxies.ProductAnalytics.ProductAnalyticsProxy  
publicDictionary<string, ProductAnalytics>  
GetProductAnalyticsLog(CelloSaaS.Model.ISearchCondition condition)
```

The sample data can be displayed in the grid as shown below:

CelloSaaS Product Analytics Audit					
User Name	URL	Time Stamp	IP Address	Method	Mode
anand@tps.com	http://localhost:58707/User/ManageUser	04-Nov-2011	::1	GET	Web
anand@tps.com	http://localhost:58707/Roles/RoleDetailsList	04-Nov-2011	::1	GET	Web
anand@tps.com	http://localhost:58707/Roles/AddRoleDetails? _=1320387450859	04-Nov-2011	::1	GET	AJAX
anand@tps.com	http://localhost:58707/Roles/ManagePrivilege? roleId=GR%24test_role	04-Nov-2011	::1	GET	Web

Figure 17-14 – CelloSaaS Product Analytics Audit

18 OBJECT-RELATIONAL MAPPING

ObjectRelationalMapping (ORM) allows one to cleanly apply object-oriented design, analysis, and programming techniques while hiding the specifics of dealing with the relational system.

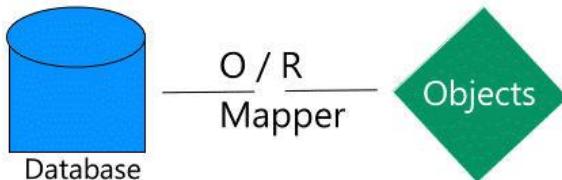
Object-relational mapping (ORM, O/RM, and O/R mapping) is a technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a "virtual object database".

Object relational mapping necessitates a bridge between application architects and database schema designers, who speak and think in different languages and have different objectives.

18.1 Entity Framework

18.1.1 Introduction

Entity Framework is an object-relational mapper (ORM) that enables .NET developers to work with relational data using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write. Entity Framework works as a layer that sits between your code and your database.



Entity framework provides complete support to O-R mapping. Entity framework is perfect to build enterprise-wide DAL because of the support for host of data providers. It provides high flexibility to adapt to specific scenarios. For multi tenancy database scenarios, writing and managing database specific data access constructs is highly challenging. Entity framework solves this problem by implementing an Object Relational (OR) interface on top of existing Relational or non-relational data sources. Entity Framework makes this possible by building a logical abstraction layer called Entity Data Model (EDM) which hosts mapping between objects and relational table. Entity framework supports connectivity to multiple databases. Basically the Entity Data Models that are built using the Entity Framework are going to be the centerpiece of communications with several technologies.

18.1.2 Why do we need Entity framework?

Entity framework is an Object/Relational Mapping (O/RM) framework. ORM is a tool for storing data from domain objects to relational database like MS SQL Server in automated way without much programming. O/RM includes three main parts: Domain class objects, Relational database objects and Mapping information on how domain objects maps to relational database objects (tables, views & storedprocedures). ORM helps us to keep our database design separate from our domain class design. This makes application maintainable and extendable. It also automates standard CRUD operation (Create, Read, Update & Delete) so developer doesn't need to write it manually.

18.1.3 Purpose of Entity framework

- Raising the level of abstraction from the logical (relational) level to the conceptual (entity) level
- Building enterprise-scale and enterprise-wide DAL needing connectivity to multiple heterogeneous databases.
- Building database-independent applications/products so that DAL can easily plugged in and out from one database to another.
- Applications having very strong domain model and needing object relational mapping.

18.1.4 Entity Framework – Key Features

Conceptual approach to enterprise development

- High performance
- Wide support for LINQ to Entities
- Wide support for EntitySQL
- Full CRUD (Create, Retrieve, Update, Delete) support
- Wide support for server data types
- Change tracking support
- Object data caching
- Inheritance mapping support (TPH, TPT, and TPCT models)
- Capability to use stored procedures when manipulating data
- Stored procedure import support
- Ability to build database-independent applications
- Visual Studio EDM Wizard support
- Reverse engineering of database objects to entity model

18.1.5 Entity Framework – Approach

Entity framework supports the development of applications using the three popular ways:

- Examine the entity classes themselves and derive an EDM from those classes – the "Code First" approach.
- Derive it by examining an existing database – the "Data First" approach.
- Write a purely conceptual entity model and generate an EDM from that – the "Model First" approach.

18.1.6 Entity framework – Code First

Code First is a style of Entity Framework model development in which you write the entity classes and map them to the database by hand. In Code First approach, you avoid working with visual model designer (EDMX) completely. You write your POCO classes first and then create database from these POCO classes. Developers who follow the path of Domain-Driven Design (DDD) principles prefer to begin by coding their classes first and then generating the database required to persist their data.

One important thing to understand is that there are two new types introduced for Code First approach, DbContext and DbSet. DbContext is a simplified alternative to ObjectContext and is the primary object for interacting with a database using a specific model. DbSet (Of TEntity) is a simplified alternative to ObjectSet(Of TEntity) and is used to perform CRUD operations against a specific type from the model in Code First approach.

Code-First Development enables a pretty sweet development workflow. It enables you to:

- Develop without ever having to open a designer or define an XML mapping file
- Define your model objects by simply writing “plain old classes” with no base classes required
- Use “convention over configuration” approach that enables database persistence without explicitly configuring anything
- Optionally override the convention-based persistence and use a fluent code API to fully customize the persistence mapping

18.1.7 Entity framework – Model First

In Model First approach, you create Entities, relationships, and inheritance hierarchies directly on the design surface of EDMX. So in Model First approach, when you add ADO.NET Entity Data Model, you should select ‘Empty Model’ instead of ‘Generate from database’.

In Model First approach you manually draw conceptual entities and their associations on the design canvas, configuring their attributes in the “Properties” panel, and (optionally) generating the Data Definition Language (DDL) script to create the database from this conceptual entity model. "Model First" works well when you are starting fresh and prefer to explore entity model possibilities unencumbered by a database schema. You don't have to generate the corresponding database definition until you are ready.

18.1.8 CelloSaaS integration with Entity Framework:

CelloSaaS supports three essential styles of entity model development:

- Generate the entity class model and maintain it with the Visual Studio Entity Data Model (EDM) Designer. – Model first
- Write the entity classes and map them to the database by hand, entirely in Code First
- Write the entity class model (and often the data access layer as well) using a technology other than Entity Framework. – Data First

CelloSaaS considers 'Code First' approach in building multi-tenant application. Code-first is a really great feature for developers, letting to focus on the data objects in an application rather than on the database or the Entity Data Model. In fact, a code-first application won't even have a model, an .edmx file in the project, or any XML mapping code. Entity Framework takes care of inferring the model, including the conceptual, storage, and mapping models, at runtime. It's really a code-centric style of writing database applications.

CelloSaaS is specially suited for many levels of the enterprise application architecture stack, its Data Services, Object Relational Mapping (ORM) technologies, and object-oriented approach to data management. CelloSaaS delegates to the Entity Framework the mapping between object and relational database schemas, as well as the database persistence operations (queries and saves). These are important and challenging tasks that the CelloSaaS Entity Framework handles well.

The Entity Framework (EF) is a core technology in most CelloSaaS Applications. You will probably define your Entity Data Model with EF's design tool and rely on EF to query and save data. We'll show you how to use the EF designer to define CelloSaaS entities in the Model.

18.1.9 Extended field support in Entity Framework

CelloSaaS builds on the strengths of the Entity Framework, and then extends them with n-tier architecture, multiple data sources, non-relational data sources, and superior client application performance.

18.1.10 Implementing Entity Framework in CelloSaaS

CelloSaaS Entity Framework maps relational tables, columns, and foreign key constraints in logical models to entities and relationships in conceptual models.

☞ Step: 1 creating the Entity Data Model, Create a project file i.e, EFService. Right click the project (EFService) and select the "Add/NewItem option

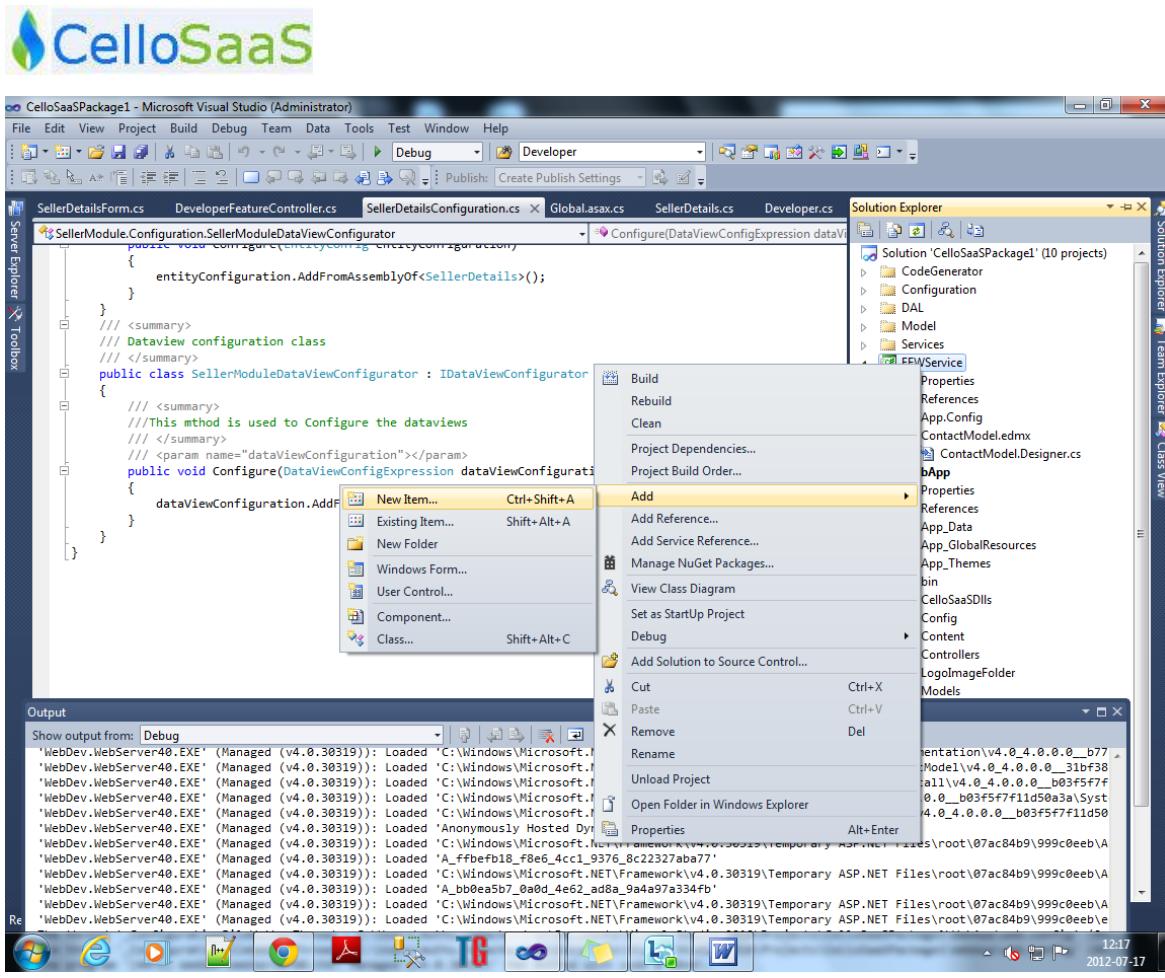


Figure 18-1 – Add/NewItem option

☞ Step 2: This will bring up the “Add New Item” dialog shown in the image below. Select the “ADO.NET Entity Data Model”, template and name the model as EmployeeModel and click the “Add” button at the bottom of the form.

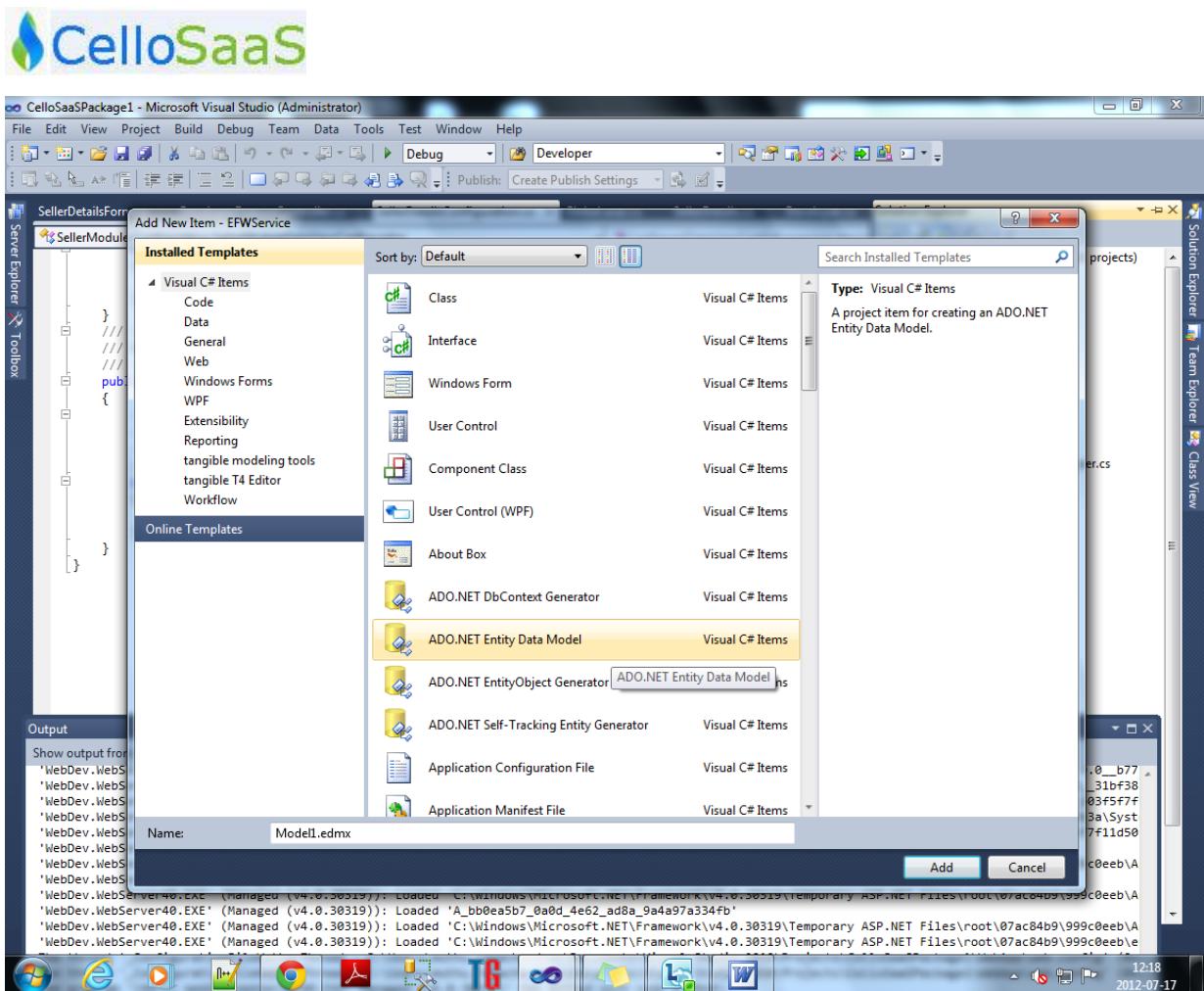


Figure 18-2 – ADO.NET DbContent Generator

- ☞ Step 3: This will bring up the “Entity Data Model Wizard” shown in the image below. The first step allows you to select whether you want to create the model from an existing database or from an empty model. Select ‘Generate from database’ option and click “Next”.

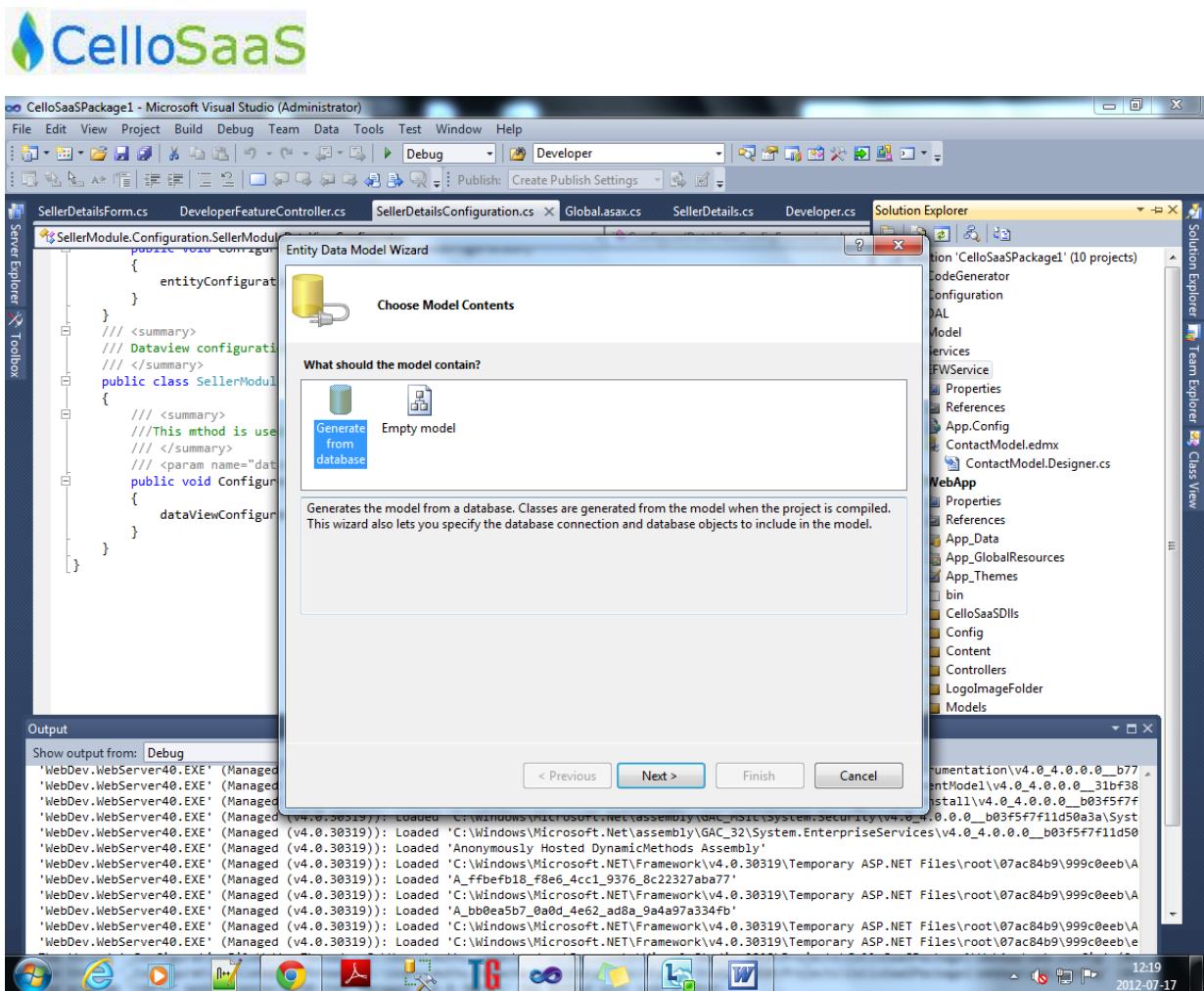


Figure 18-3 – Entity Data Model Wizard

- ☞ Step 4: You can choose the data connection from your existing DB Connections or create new connection by clicking ‘New Connection’ button. If you select the checkbox ‘Save entity connection settings in App.Config’ as: This will also add connection string to your app.config file with default suffix with db name. You can change it if you want. Click ‘Next’ after you set up your db connection.

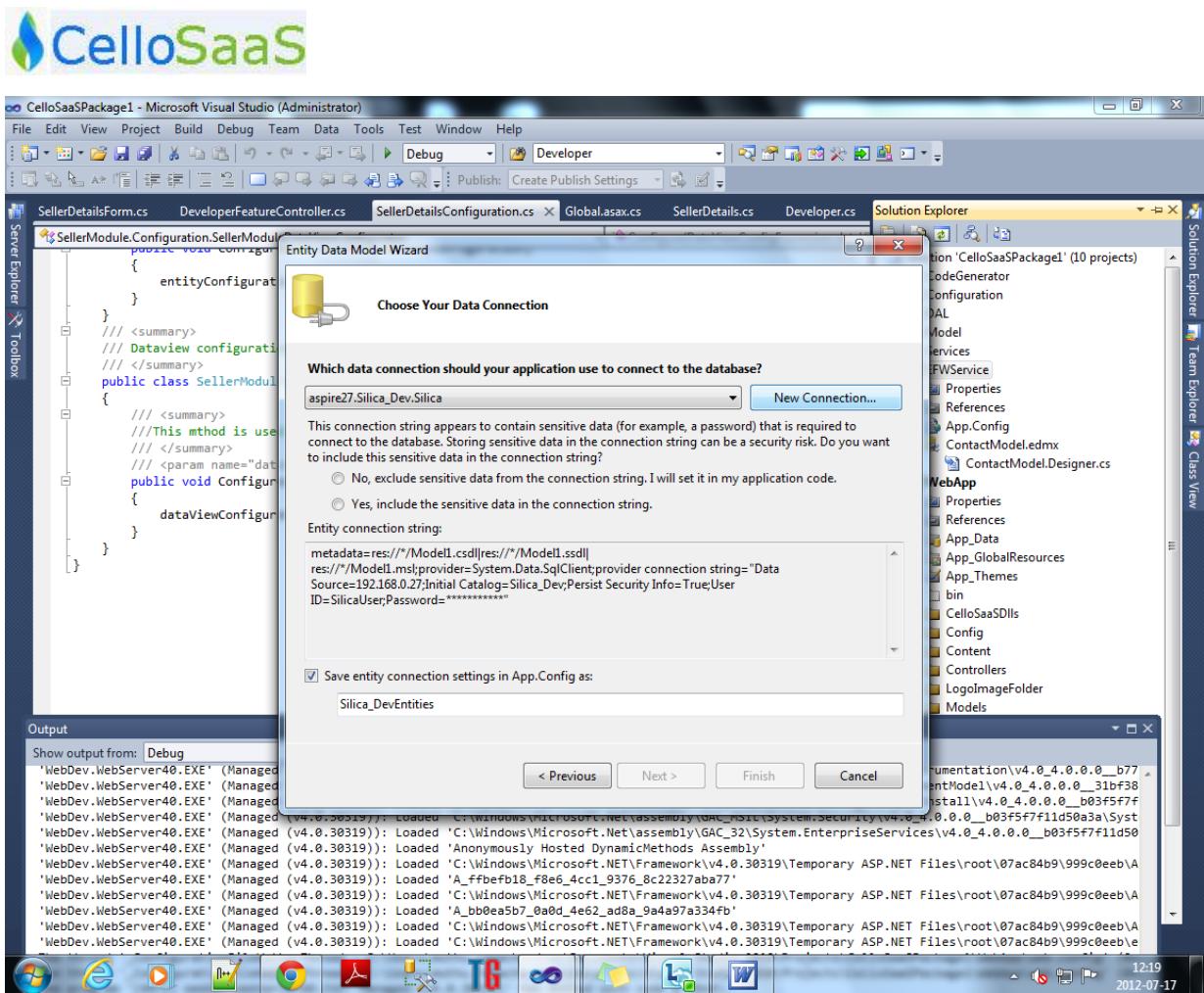


Figure 18-4 – Entity Data Model Wizard

- ☞ Step 5: This step will display all the Tables, Views and Stored Procedures in the database. Select tables, views and SPs you want and click ‘Finish’. Name the connectionstring and name entity connection settings as EmployeeEntities.

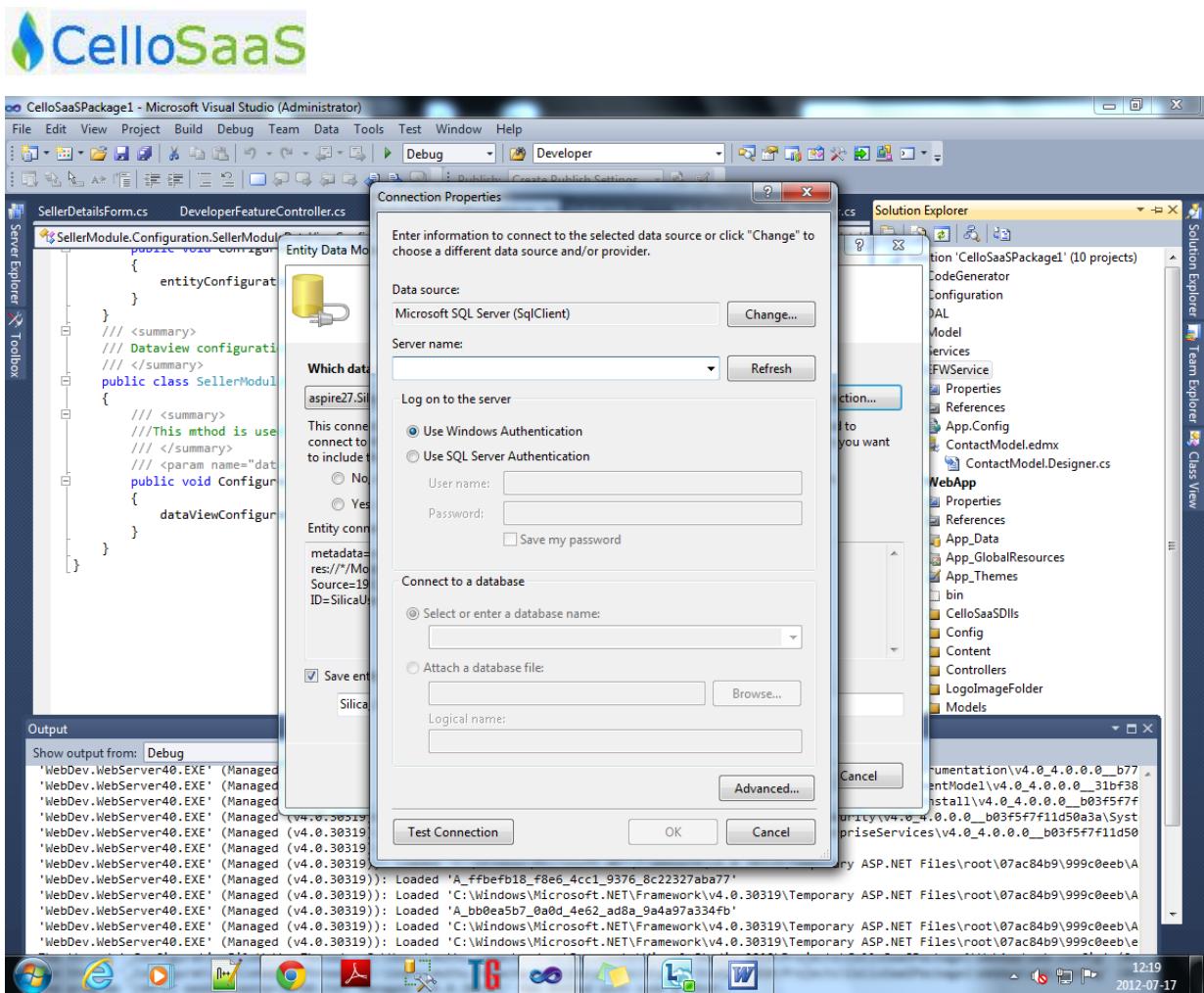


Figure 18-5 – Connection Properties

☞ Step 6: The wizard populates the employee model edmx file and app.config file

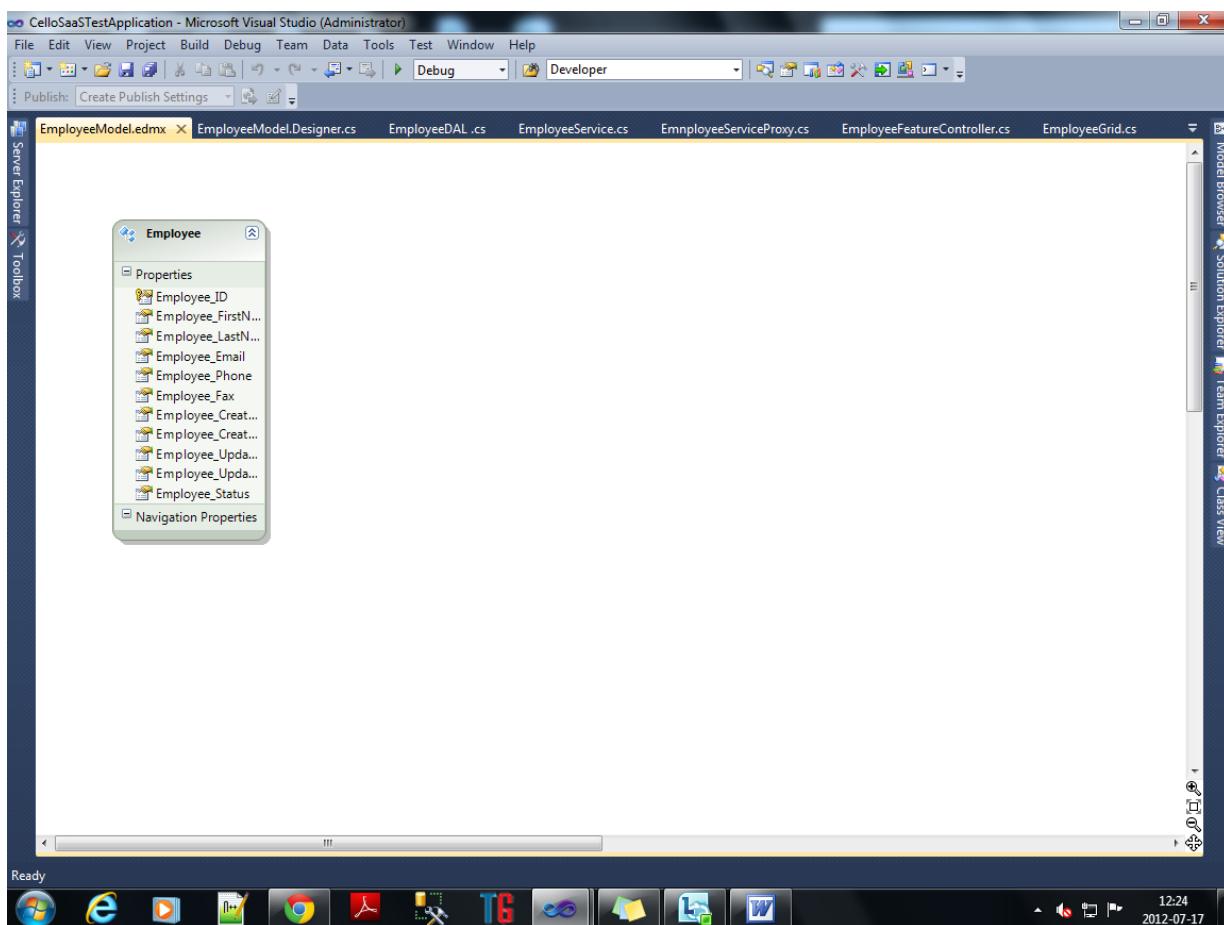


Figure 18-6 – Connection Properties

- ☞ Step 7: In the property of the model, set the property Code Generation strategy to none. and open designer.cs. Customize the code and add reference to the model. Add another class and name it as a EmployeeEntities and write the following code.

```
using System.Data.Objects;
using EmployeeModule.Model;
public class EmployeeEntities : ObjectContext
{
    #region ObjectSet Properties
    public EmployeeEntities()
        : base("name=EmployeeEntities", "EmployeeEntities")
    {
        _Employees = CreateObjectSet<Employee>();
    }
    ///<summary>
    /// No Metadata Documentation available.
    ///</summary>
    public ObjectSet<Employee> Employees
    {
        get
        {
            if (_Employees == null)
            {
                _Employees = base.CreateObjectSet<Employee>("Employees");
            }
        }
    }
}
```

```

        return _Employees;
    }
}
privateObjectSet<Employee> _Employees;

#endregion
}

```

☞ Step 8: You can get the overall code from the code generation. Replace the DAL with following changes.

```

Ref – system.data.entity
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using CelloSaaS.DAL;
using CelloSaaS.Library.DataAccessLayer;
using System.Data.Common;
using System.Data;
using EmployeeModule.Model;
using EmployeeModule.DAL.EmployeeFeature;
///<summary>
/// Look into this to implement SqIDAL class.
///</summary>
namespace EmployeeModule.SqIDAL.EmployeeFeature
{
///<summary>
/// This is the data implementation class for EmployeeDAL
/// EntityDAL with entity type name has to be inherited.
/// This class must inherit same name of Interface.
/// CRUD operation of EmployeeDAL will be implemented in this class.
///</summary>
public class EmployeeDAL : EntityDAL<Employee>, IEmployeeDAL
{
///<summary>
/// Add method for the employee entity
///</summary>
///<param name="dataCreateRequest"></param>
///<returns></returns>
protected override string DoCreate(DataCreateRequest dataCreateRequest)
{
EmployeeEntities context = new EmployeeEntities();
Employee employee = (Employee)dataCreateRequest.Entity;
employee.Employee_ID = Guid.NewGuid();
employee.Identifier = employee.Employee_ID.ToString();
context.AddObject("Employees", employee);
context.SaveChanges();
return employee.Identifier;
}
///<summary>
/// delete method for employee entity
///</summary>
///<param name="dataDeleteRequest"></param>
protected override void DoDelete(DataDeleteRequest dataDeleteRequest)
{
EmployeeEntities context = new EmployeeEntities();
List<Employee> employeeList = context.Employees.ToList();

```

```

var employee = from a in employeeList
where a.Employee_ID.ToString().ToUpper() == dataDeleteRequest.Identifier.ToString().ToUpper()
select a;
Employee emp = newEmployee();
    emp = employee.ToList()[0];
    context.Employees.DeleteObject(emp);
    context.SaveChanges();
}
///<summary>
/// fetch method for employee entity
///</summary>
///<param name="dataFetchRequest"></param>
///<returns></returns>
protectedoverrideEmployee DoFetch(DataFetchRequest dataFetchRequest)
{
EmployeeEntities context = newEmployeeEntities();
List<Employee> employeeList = context.Employees.ToList();
var employee = from a in employeeList
where a.Employee_ID.ToString().ToUpper() == dataFetchRequest.Identifier.ToString().ToUpper()
select a;
    employee.ToList()[0].Identifier = dataFetchRequest.Identifier;
return employee.ToList()[0];
}
///<summary>
/// Search method for employee entity
///</summary>
///<param name="dataSearchRequest"></param>
///<returns></returns>
protectedoverrideDictionary<string, Employee> DoSearch(DataSearchRequest dataSearchRequest)
{
EmployeeEntities context = newEmployeeEntities();
List<Employee> employeeList = context.Employees.ToList();
Dictionary<string, Employee> employeeDetails = newDictionary<string, Employee>();
if (dataSearchRequest != null&& dataSearchRequest.Identifiers != null)
{
var employee = from a in employeeList
where a.Employee_ID.ToString().ToUpper().Contains(dataSearchRequest.Identifiers.ToString())
select a;

foreach (var employees in employee)
{
    employees.Identifier = employees.Employee_ID.ToString();
if (!employeeDetails.ContainsKey(employees.Employee_ID.ToString()))
        employeeDetails.Add(employees.Employee_ID.ToString(), employees);
}
else
{
foreach (var employees in employeeList)
{
    employees.Identifier = employees.Employee_ID.ToString();
if (!employeeDetails.ContainsKey(employees.Employee_ID.ToString()))
        employeeDetails.Add(employees.Employee_ID.ToString(), employees);
}
}
return employeeDetails;
}
///<summary>

```



```
/// Update method for Employee Entity
///</summary>
///<param name="dataUpdateRequest"></param>
protected override void DoUpdate(DataUpdateRequest dataUpdateRequest)
{
    EmployeeEntities context = new EmployeeEntities();
    Employee employee = (Employee)dataUpdateRequest.Entity;
    List<Employee> employeeList = context.Employees.ToList();
    Employee employees = employeeList.First(i => i.Employee_ID.ToString().ToUpper() == employee.Identifier.ToUpper());
    employees.Employee_FirstName = employee.Employee_FirstName;
    employees.Employee_LastName = employee.Employee_LastName;
    employees.Employee_Phone = employee.Employee_Phone;
    employees.Employee_Fax = employee.Employee_Fax;
    employees.Employee_Email = employee.Employee_Email;
    employees.Employee_UpdatedBy = employee.Employee_UpdatedBy;
    employees.Employee_UpdatedOn = employee.Employee_UpdatedOn;
    context.SaveChanges();
}
```

- ☞ Step 9: Then remove the code from update method in controller
employee.Employee_ID=employee.Identifier;
 - ☞ Step 10 : In model you have to change the string to guid return type for primary key and other guid fields. Then rebuild the application and you can use the CelloSaaS feature.

Methods

Entity framework connectionString

This method is used to get an entity framework connectionstring and set it in entity connection and to get the datas from appropriate database.

```
///<summary>
/// Get the entity frame work connectionstring based on connection string name
///</summary>
///<param name="connectionStringName">connectionStringName</param>
///<returns>ConnectionString</returns>
public string GetEntityFrameWorkConnectionString(string connectionStringName)
```

Example

```
EntityObjectModel context = newEntityObjectModel();
EntityConnection conn = newEntityConnection();
    conn.ConnectionString =
EntityFrameWorkProxy.GetEntityFrameWorkConnectionString("EntityFrameWorkconnectionString");
context = newEntityObjectModel(conn.ConnectionString);
```

Datascope

The method is used to get the datascope filtered data for the particular entity. Pass the entityid and permission name as parameters and get the list of ids of the corresponding entity and then check the ids and return the datas.

```
///<summary>
/// Get the datascope filter data based on entityId and permissionName
///</summary>
///<param name="entityId">entityId</param>
///<param name="permissionName">permissionName</param>
```

```
///<returns>List<string></returns>
publicList<string> GetDataScopeFilterData(string entityId, string permissionName)
```

Example

```
EntityObjectModel context = newEntityObjectModel();
List<Employee> employeeLists = context.Employees.ToList();
List<string> ids = DataAccessProxy.GetDataScopeFilterData("EmployeeDetails",
"View_EmployeeDetails");
var employeeList = from emp in employeeList
where ids.Contains(emp.EmployeeID.ToString())
select emp;
```

Tenant Scope

The method is used to apply the entity tenant scope, pass the entity id and fetch type to this method and gets the list of tenantids and then checks whether the ids available then return the list.

```
///<summary>
/// Get the List of tenant ids for the entity tenant scope
///</summary>
///<param name="entityId">entityId</param>
///<param name="fetchType">fetchType</param>
///<returns>List<tenantids></returns>
publicList<string> GetAccessedTenants(string entityId, FetchType fetchType)
```

Example

```
EntityObjectModel context = newEntityObjectModel();
List<TenantDetail> tenantList = context.TenantDetails.ToList();
FetchType fetchType = FetchType.Edit;
List<string> tenantids = TenantAccessProxy.GetAccessedTenants("User", fetchType);
var tenant = from tenantid in tenantList
where tenantids.Contains(tenantid.Tenant_Code.ToString())
select tenantid;
```

19 HOSTING YOUR SAAS APPLICATION

For hosting your application, you have following options:

As Cellosaas is based on standard .net environment, you can host it like any other .net applications. You can host it on a dedicated server or on a public cloud.such as amazon, rackspace, etc.

19.1 Deploying CelloSaaS in Windows Azure

- ☞ Step 1: In the CelloSaaS solution, add New Project “Windows Azure Project”.
- ☞ Step 2: Press Ok button. Avoid selecting anything in the pop-up window.

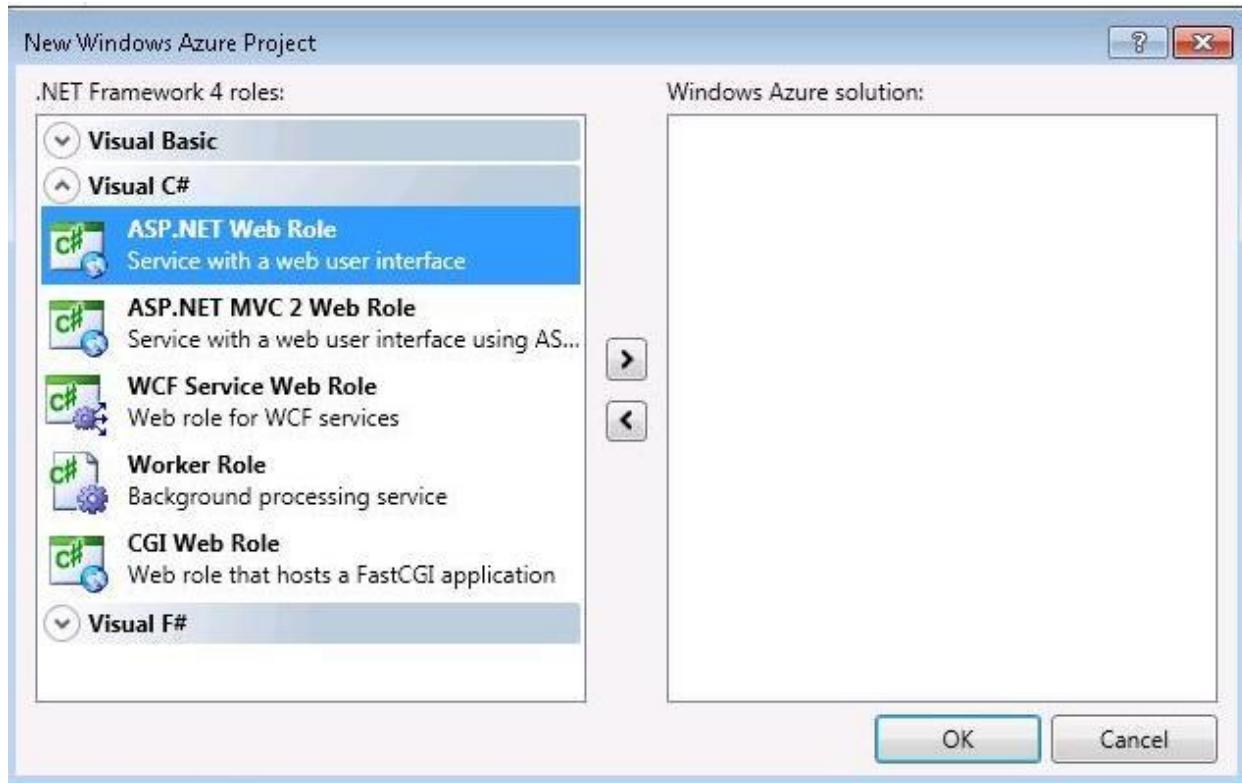


Figure 19-1 – New Windows Azure Project

- ☞ Step 3: Right Click “Roles” Add Web Role Project In Solution.
- ☞
- ☞
- ☞
- ☞
- ☞
- ☞
- ☞
- ☞

19.2 Scalability in CelloSaaS

CelloSaaS can be scaled out either by scaling up or scaling out. Scaleup is straightforward. Let us see how the different layers of the CelloSaaS application can be individually scaled out.

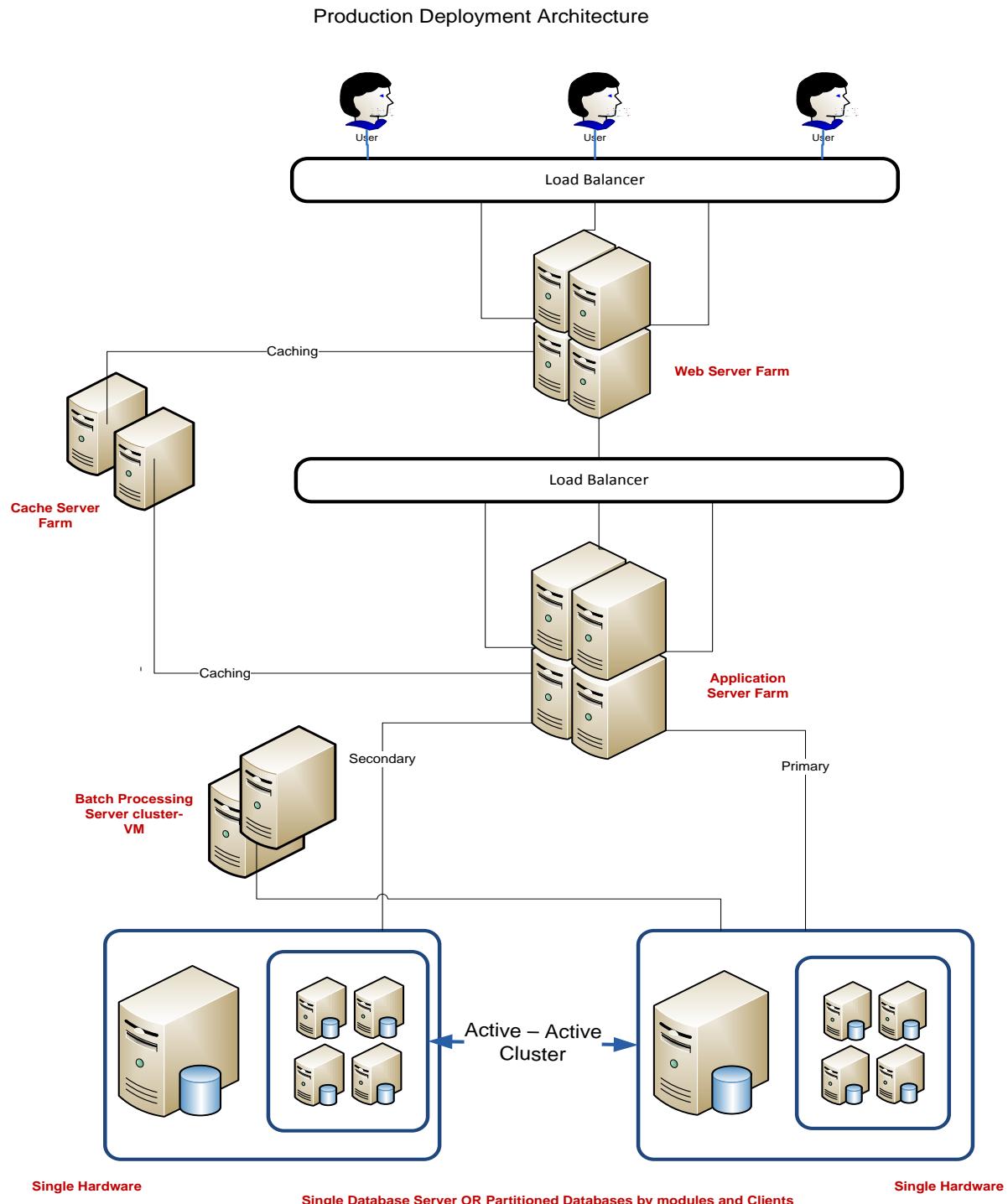


Figure 19-2 – Production Deployment Architecture Diagram.

Web Layer: This is responsible for rendering the user interface. CelloSaaS advocates the following principle to **scale out** this layer.



Session Usage: CelloSaaS does not use session for storing any data. If the application needs to store session cellosaas mandated outofproc session storage. This ensure that the application is stateless and hence can be easily scaled out

Cache Usage: CelloSaaS uses Appfabric distributed cache as the caching layer. This ensures that the memory state of cache is centralized and hence the application becomes stateless which is necessary to be scaled out.

Application Layer- This is responsible for the web service driven business layer. CelloSaaS advocates the following principle to scale out this layer

Per Call Services - All services are per call instances and hence can be scaled infinitely

Cache Usage - Application Layer also uses centralized caching mechanism of Appfabric which ensures that the application layer is stateless and hence can be scaled out.

Cache Layer: CelloSaaS uses Distributed cache – appfabric. This ensures that the cache layer can be scaled out by adding more nodes if there is a higher memory requirement for cache.

Database Layer: CelloSaaS supports database sharding by Tenant+module combination. Data of a module+tenant can be partitioned and can stay in different databases. For example Module1 data of Tenant 1 and 2 can be on DBServer1. Module 2 data of tenant 1 can be on DBServer2 and Module 2 of tenant 2 can be on DBSever 3.

19.3 App Fabric Installation

- ☞ Step 1: Download AppFabric from the following link based on caching deployment machine.

Platform	Setup package
Windows Vista and Windows Server 2008 x64	WindowsServerAppFabricSetup_x64_6.0.exe
Windows 7 and Windows Server 2008 R2 x64	WindowsServerAppFabricSetup_x64_6.1.exe
Windows Vista and Windows Server 2008 x86	WindowsServerAppFabricSetup_x86_6.0.exe
Windows 7 x86	WindowsServerAppFabricSetup_x86_6.1.exe

- ☞ Step 2: Download Link

<http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=15848>

- ☞ Step 3: Accept the licence agreement and click “Next>”.

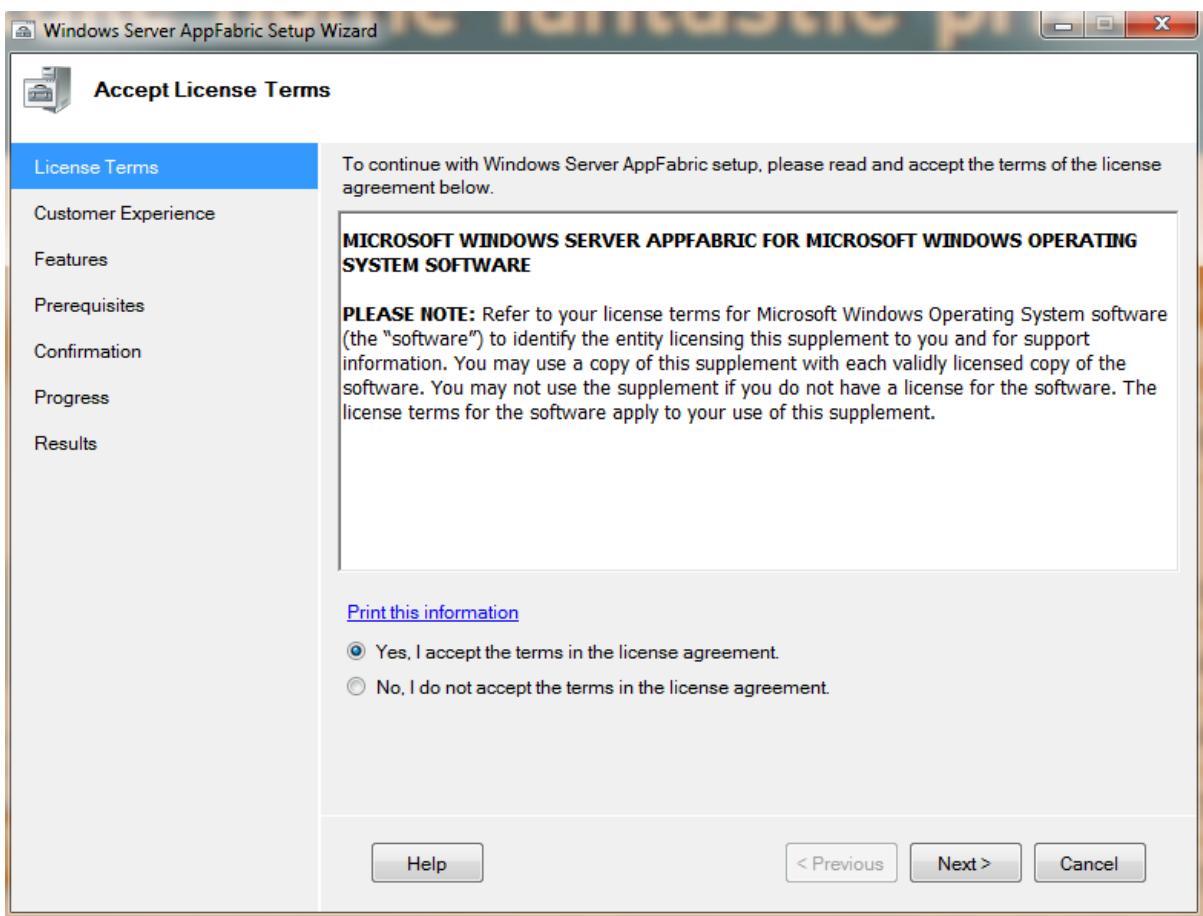


Figure 19-3 – Accept License Terms

☞ Step 4: Click “Next>”.

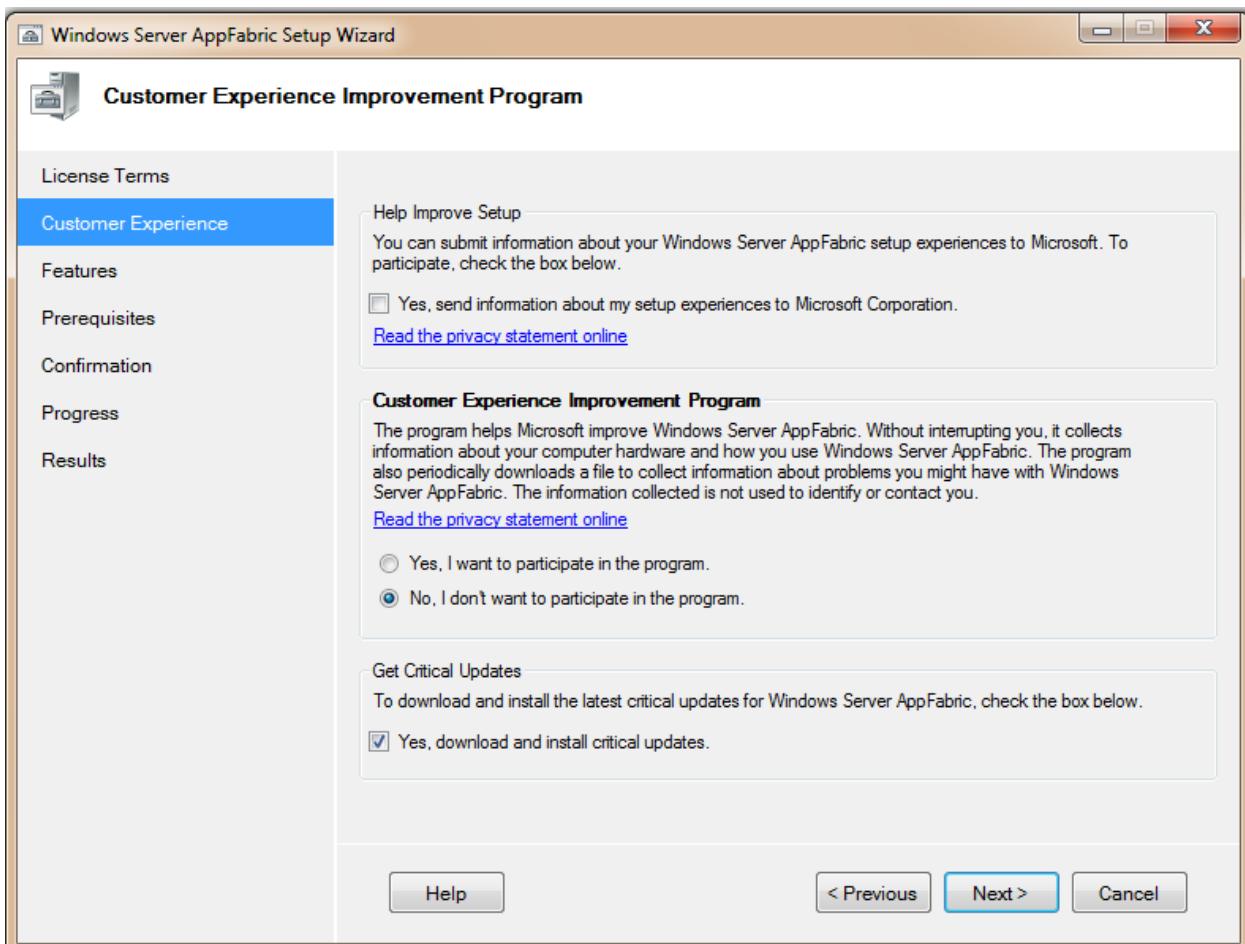


Figure 19-4 – Customer Experience Improvement Program

☞ Step 5: Select all options and click “Next>”.

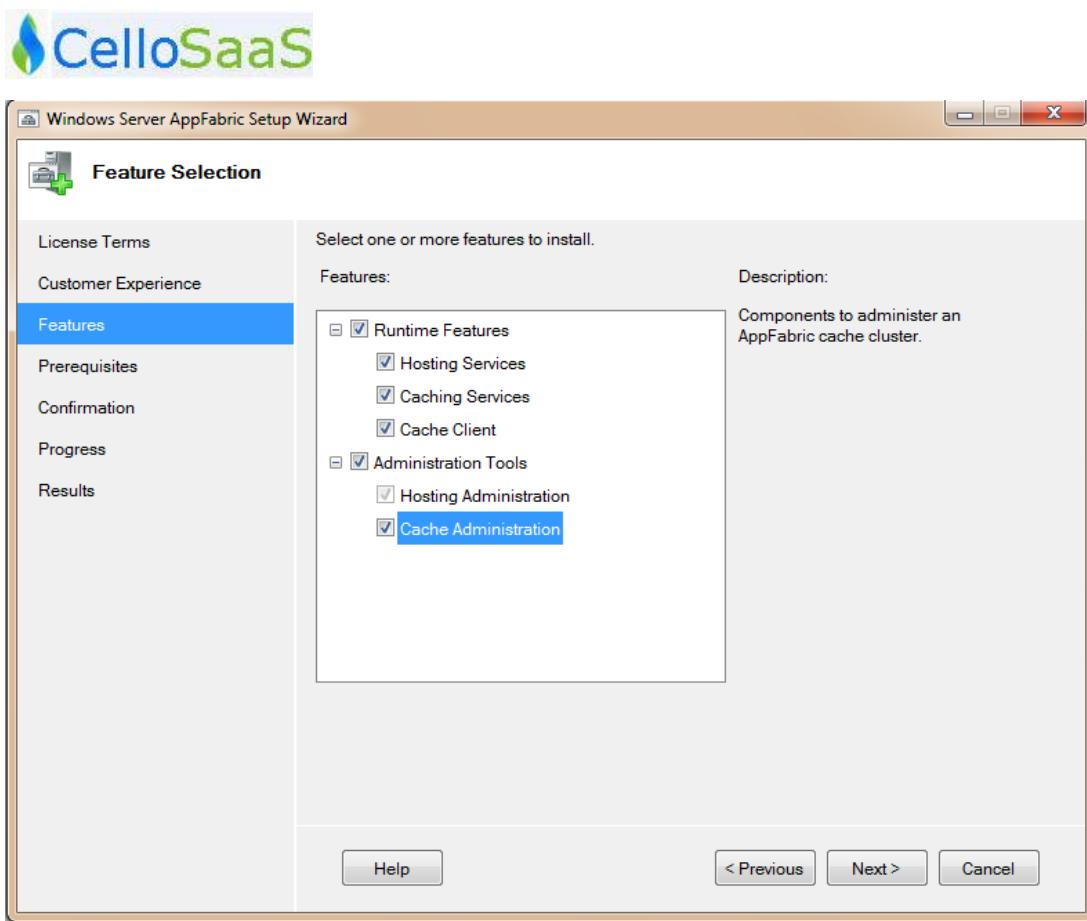


Figure 19-5 – Feature Selection

☞ Step 6: Click “Next>”.

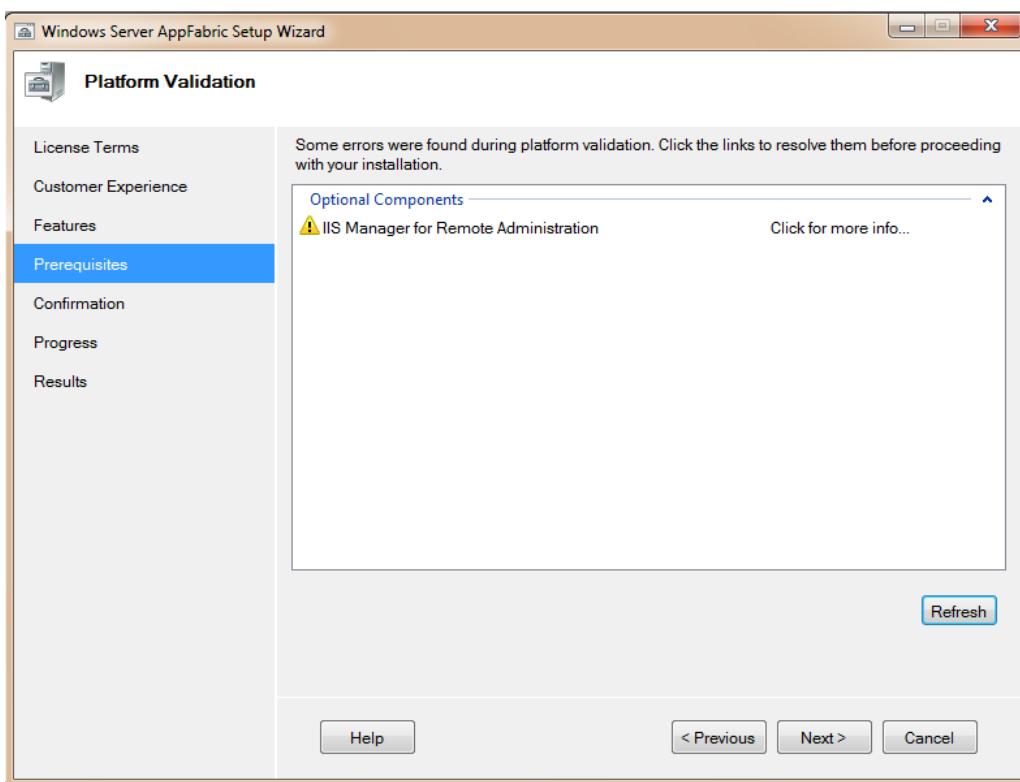


Figure 19-6 – Platform Validation

☞ Step 7: Click “Install”.

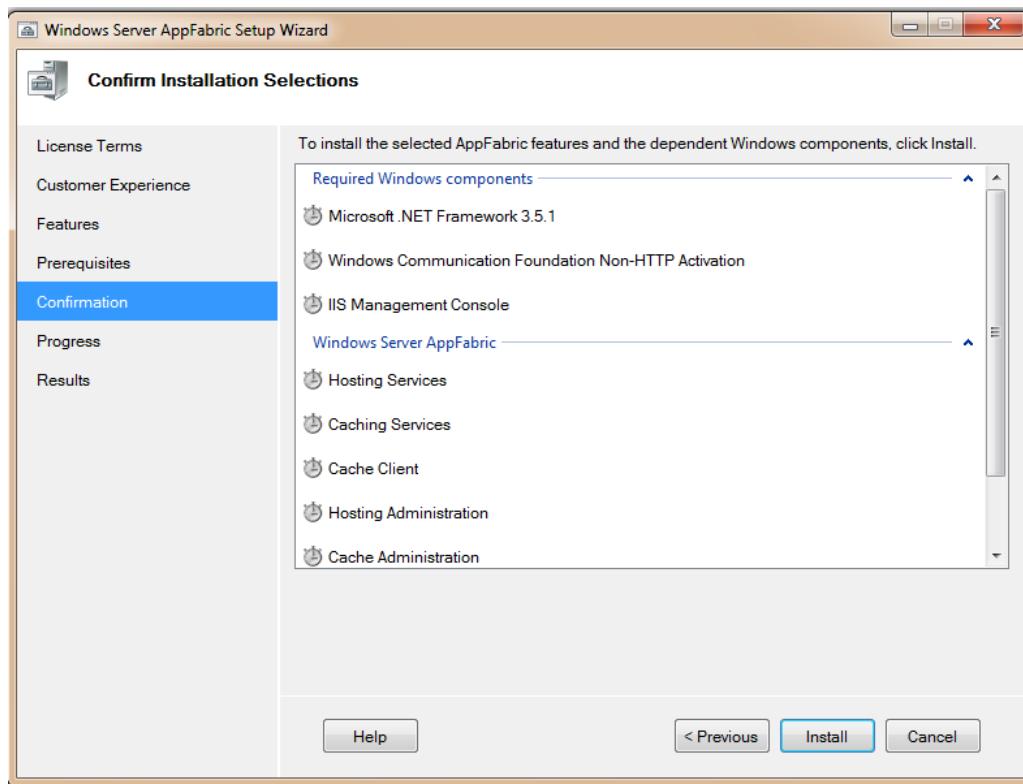


Figure 19-7 – Confirm Installation Selection

☞ Step 8: Click “Finish”. Then AppFabric Configuration Wizard will be available to configure cluster.

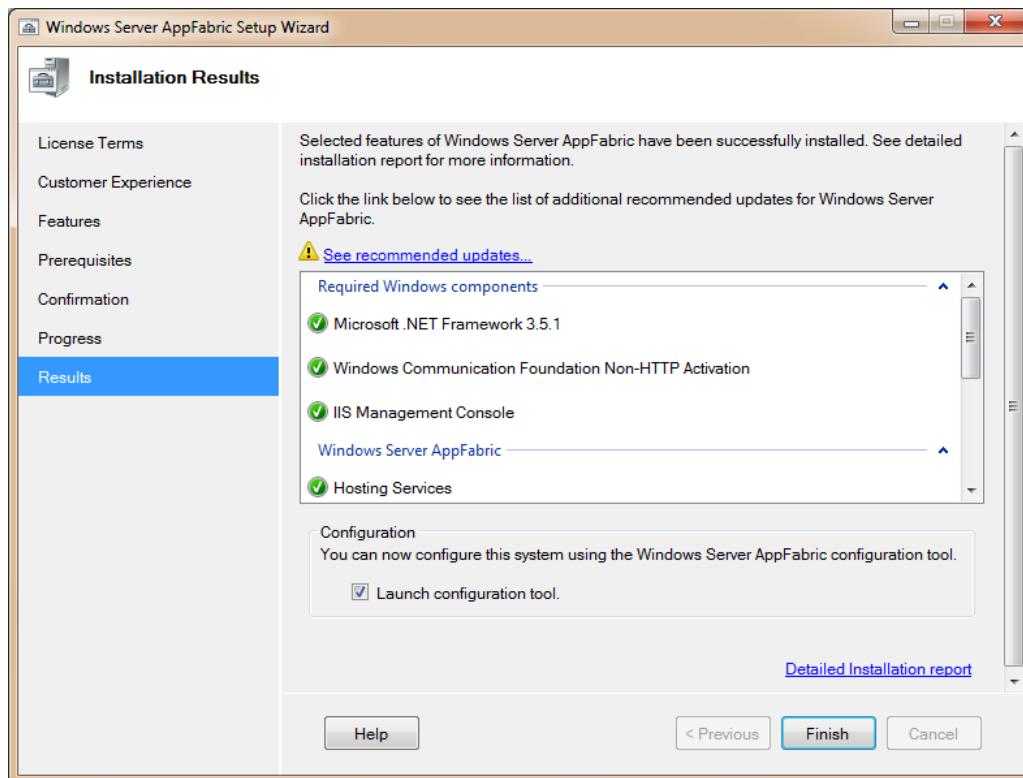
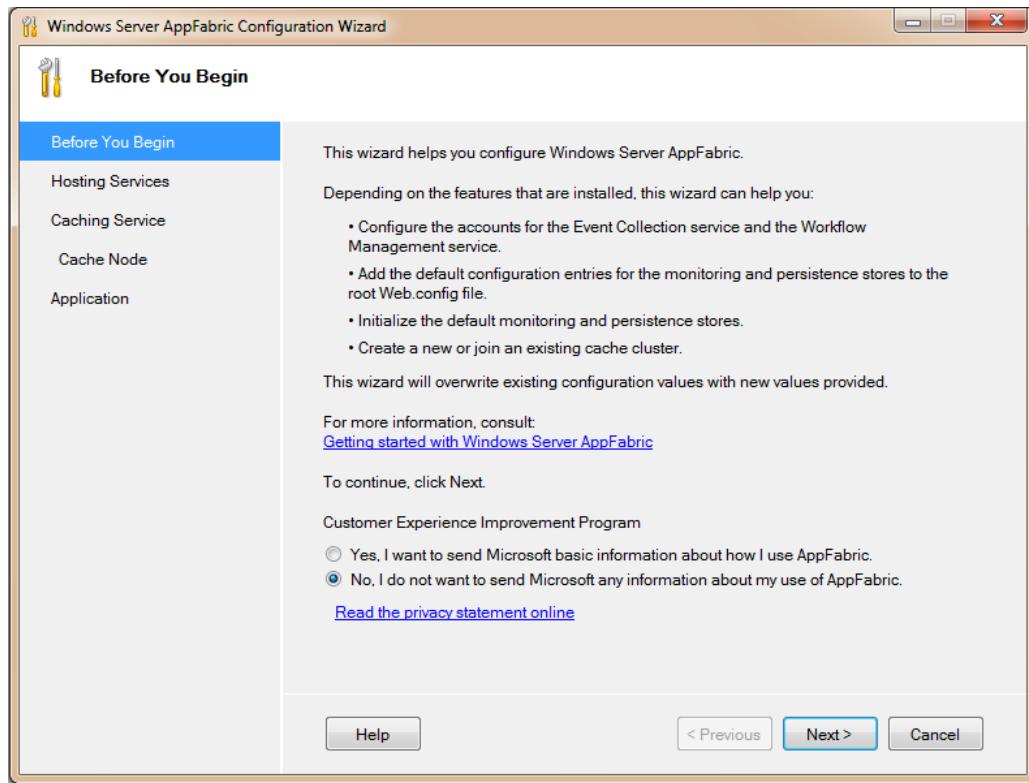


Figure 19-8 – Installation Results

☞ Step 9: Click “Next>”.

**Figure 19-9 – Before You Begin**

☞ Step 10: Click “Next>”.

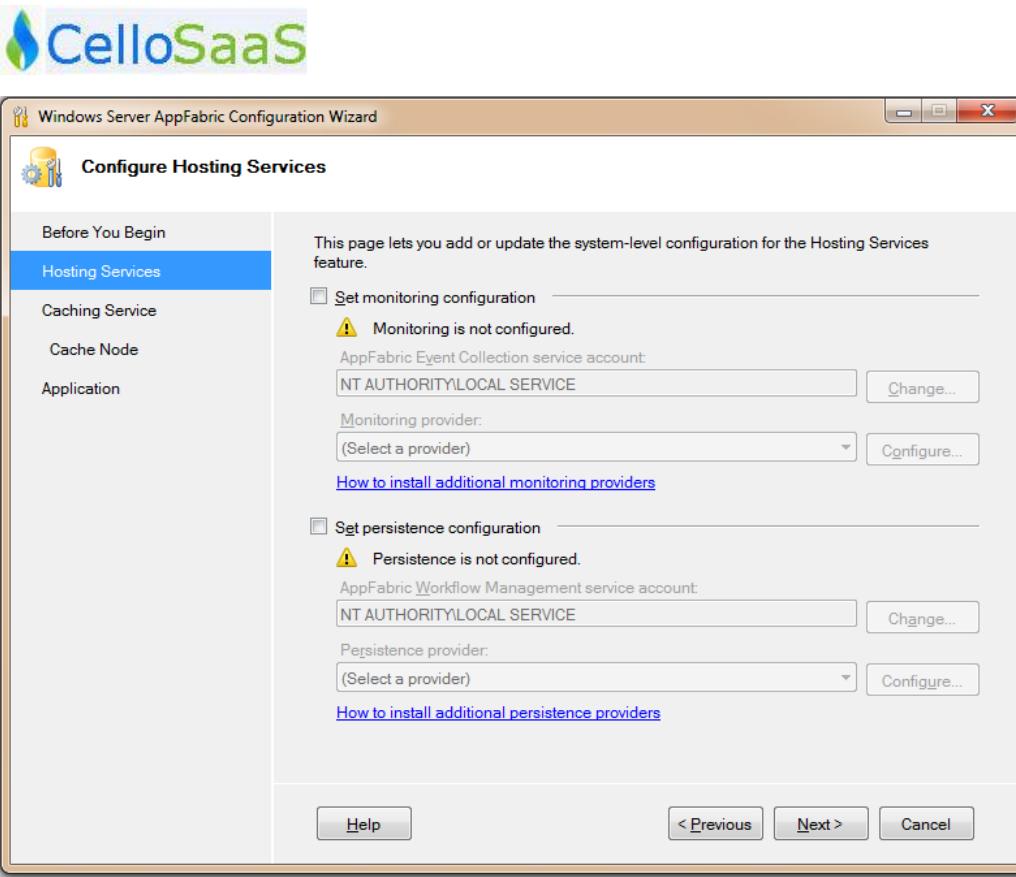


Figure 19-10 – Configure Hosting Services

☞ Step 11: Click “Next>”.

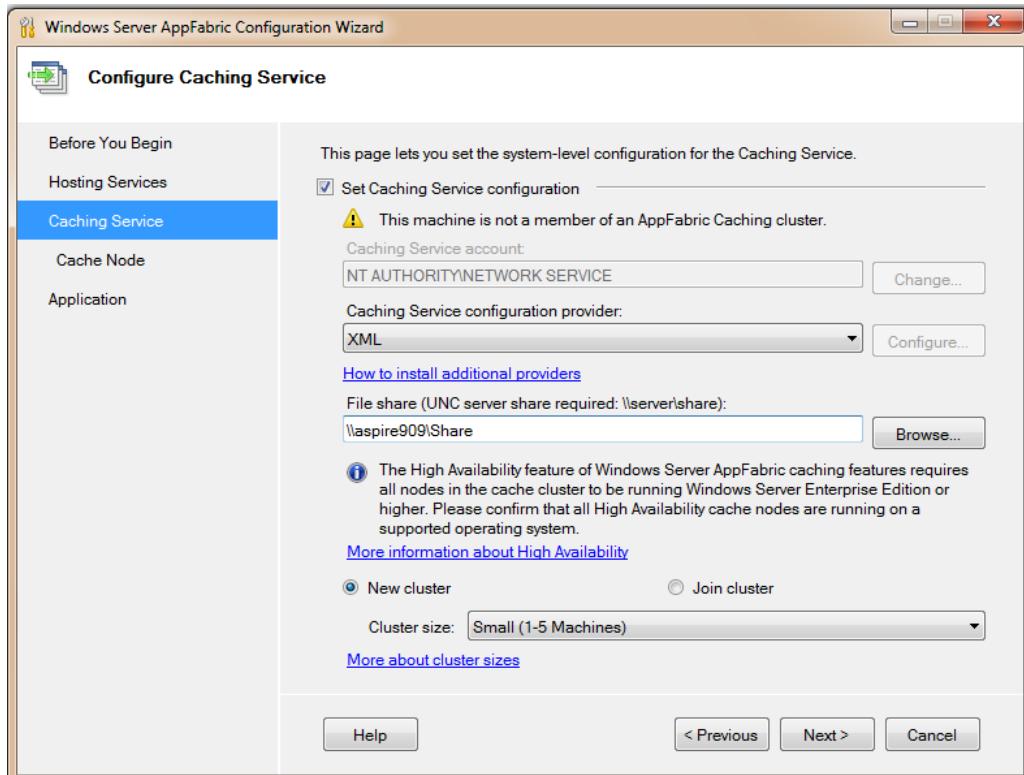


Figure 19-11 – Configure Caching Service

- ☞ Step 12: Check “Select Caching Service Configuration” and select “XML” in Caching Service Configuration provider”.
- ☞ Step 13: Give the valid shared folder name in “File Share” path.
- ☞ Step 14: Select the required Cluster Size from the dropdown(Small, Medium and Large).
- ☞ Step 15: Click “Next>”.

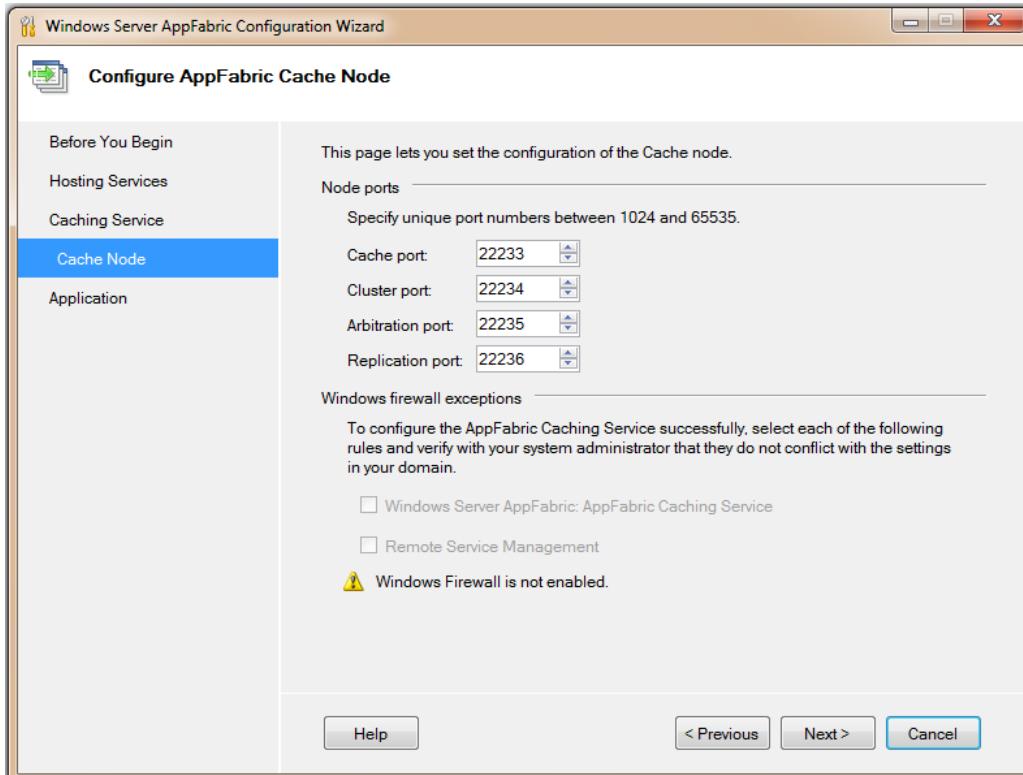


Figure 19-12 – Configure AppFabric Cache Node

- ☞ Step 16: Select “Yes”.

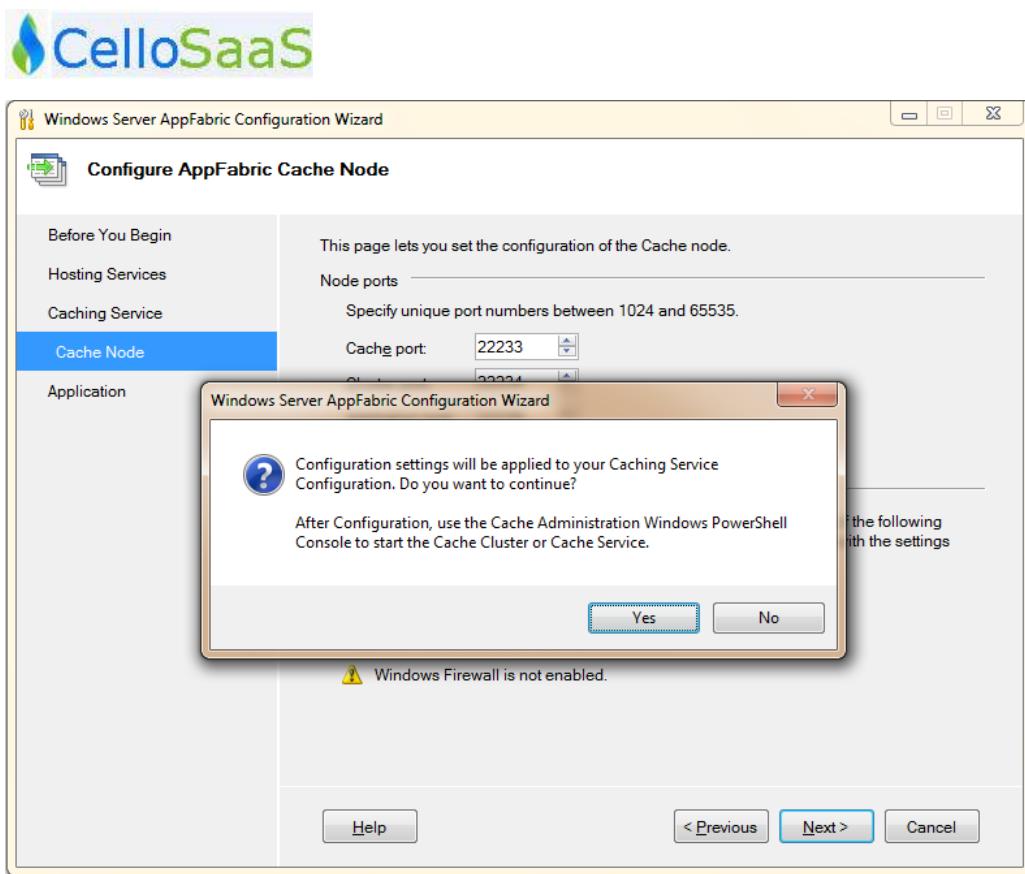


Figure 19-13 – Configure AppFabric Cache Node

☞ Step 17: Click “Finish”.

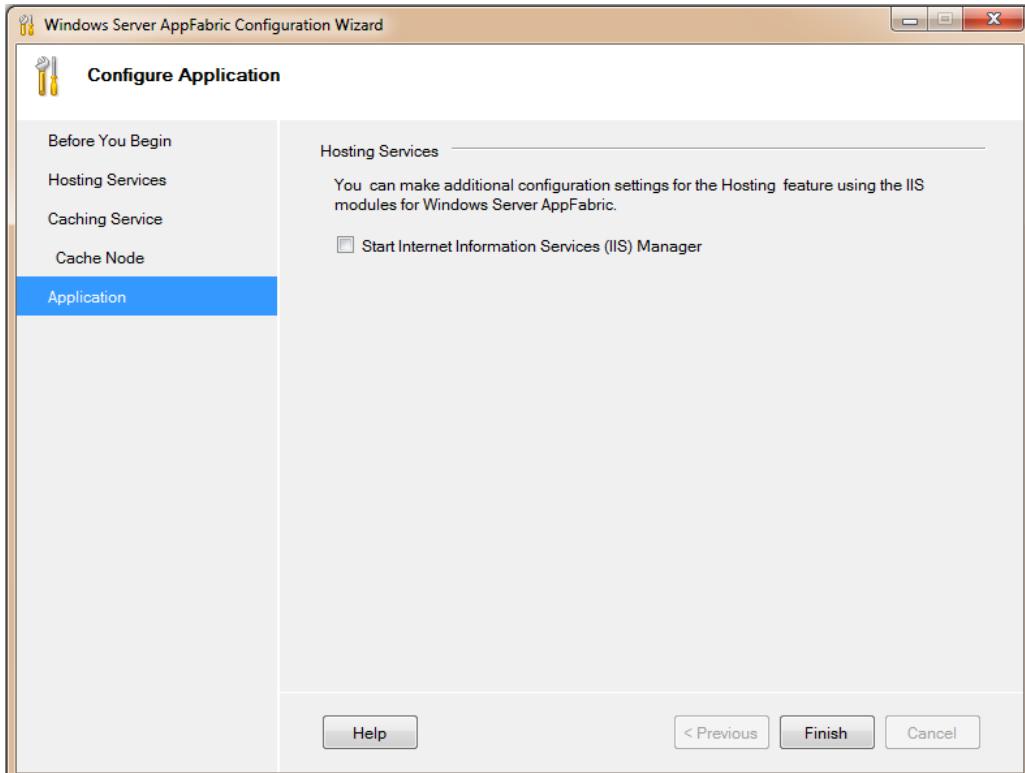
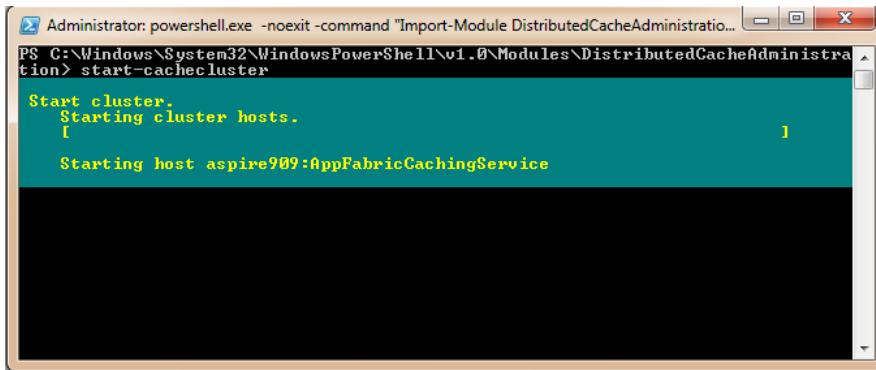


Figure 19-14 – Configure Application

- ☞ Step 18: Run Caching Administration Windows PowerShell.
- ☞ Step 19: And Start the Cache cluster using the command “**Start-CacheCluster**”.



```
Administrator: powershell.exe -noexit -command "Import-Module DistributedCacheAdministration; start-cachecluster"
PS C:\Windows\System32\WindowsPowerShell\v1.0\Modules\DistributedCacheAdministration> start-cachecluster
Start cluster.
Starting cluster hosts.
[          ] Starting host aspire909:AppFabricCachingService
```

Figure 19-15 – Start Cluster

How to use in Application

- ☞ Step 1: Add the following Dll files to the CelloSaaSDll folder if these does not exist already.
 - Microsoft.WindowsFabric.Common.dll
 - Microsoft.WindowsFabric.Data.Common.dll
 - Microsoft.ApplicationServer.Caching.Client.dll
 - Microsoft.ApplicationServer.Caching.Core.dll
- ☞ Step 2: And add references to these four dll files in the WebApplication
- ☞ Step 3: In the web.config file, add the following configuration settings for enabling the AppFabric Caching

```
<cachingConfiguration defaultCacheManager="AppFabric Cache Manager">
  <cacheManagers>
    <add name="AppFabric Cache Manager"
        type="CelloSaaS.Library.Provider.AppFabricCacheManager, CelloSaaS.Library"
        RoutingClient="false" LocalCache="false" HostName="hostname"
        CachePort="22233" CacheHostName="DistributedCacheService" NamedCache="default"
        InvalidationPolicy="TimeoutBased" DefaultTimeOut="10" SecurityMode="None"
        ProtectionLevel="None"/>
  </cacheManagers>
</cachingConfiguration>
```

- ☞ Step 4: Comment out the other cachingConfiguration in the web.config file
- ☞ Step 5: Replace the HostName attribute value with the server name in which appfabric caching is enabled
- ☞ Step 6: Appfabric Property details.

Property	Mandatory	Description	Default Value
RequestTimeout (seconds)	NO	Used to set the Client time-out. Microsoft does not recommend specifying a value less than 10000 (10 seconds).	3(15)

ChannelOpenTimeout (seconds)	NO	This attribute value can be set to 0 in order to immediately handle any network problems.	15(3)
MaxConnectionsToServer	NO	Used to set the Maximum number of connections to the server.	1
HostName	Yes	Specify the Host Name	
CachePort	Yes	Specify the Cache Port	
CacheHostName	Yes	Specify the Cache Host Name	
NamedCache	Yes	Specify the Cache Name	
LocalCache	Yes	Used to Enable the local cache. Possible values “True” and “False”.	
InvalidationPolicy	Yes	Possible values include NotificationBased and TimeoutBased.	
DefaultTimeOut (seconds)	Yes	Local cache time-out.	
ObjectCount	NO	Maximum locally-cached object count	10,000
PollInterval	NO	Specific cache notifications poll interval (seconds)	300
MaxQueueLength	NO	Maximum queue length	10,000
SecurityMode	NO	Possible values include Transport and None.	Transport
ProtectionLevel	NO	Possible values include None, Sign, and EncryptAndSign.	EncryptAndSign
ChannelInitializationTimeout (seconds)	NO	Channel initialization timeout	
ConnectionBufferSize	NO	Connection buffer size	
MaxBufferPoolSize	NO	Maximum buffer pool	
MaxBufferSize	NO	Maximum buffer size	
MaxOutputDelay(seconds)	NO	Maximum output delay	
ReceiveTimeout(seconds)	NO	Receive timeout	

20 CELLOSAAS CACHE SYNCHRONIZATION

In production scenario where we load balance the application to take on more load, we tend to deploy multiples WCF services across servers. In this case CelloSaaS recommends AppFabric Caching server. CelloSaaS also offers custom Cache Synchronization feature where cache are updated real-time using push mechanism.

Consider a scenario in which 3 applications hosted at appserver1.com, appserver2.com and appserver3.com. All of the hosted WCF services use In-Proc caching. Now configure these servers to use CelloSaaS cache synchronization feature to keep their caches up-to date.

In appserver1.com, add the below configuration in AppSettings section of the Web.config to refer to the other application servers.

```
<add key="CacheServerEndpoints" value="http://appserver2.com, http://appserver3.com"/>
```

Similarly in appserver2.com,

```
<add key="CacheServerEndpoints" value="http://appserver1.com, http://appserver3.com"/>
```

Similarly in appserver3.com,

```
<add key="CacheServerEndpoints" value="http://appserver1.com, http://appserver2.com"/>
```

You can also specify which named caches should be synchronized by CelloSaaS by using the below appSettings key.

```
<add key="CacheSyncNames" value="CelloSaaS, YourNamedCache"/>
```

To Disable Cache Synchronization give empty value for [CacheServerEndpoints](#).

Functionality

This feature from CelloSaaS will take the responsibility of synchronizing the data across the registered servers so that any data consumed from the cache in any of the server will be consistent data and data is not stale in any of these servers.

21 UTILITIES

21.1 Tenant Data backup Management

Database Backup Utility primarily for tenant Administrators and product owners to archive their mission critical data. Both CelloSaaS Metadata and Application Data can be backed up using the Administration Dashboard. The Backup utility is not a replacement or compliment for SQL Databackup, the primary purpose is to provide simple and elegant data backup solution for critical data in day to day process without much technical knowledge.

Backup Modes

None	Only tenant data (Tables with TenantId column only)
Full	Tenant data + Master data (including tables without TenantId column)

At present there is no automated process to export back to the same database or different Database. The process of uploading the backup file to a remote server or network share could be automated using schedulers. It also provides an option to backup the child tenant's data along with parent tenant's data. The backup utility employs a back up service internally to process the backup request which could run in the same process of the web server or in a separate Server.

Information such as target database instance name, username and password as well as destination details will be captured using the interface, but it is the responsibility of the administrator to make sure the given credential has full access in that particular database before doing the data backup.

The given SQL login credentials must have necessary permissions to perform the backup operations. Required SQL permissions are create database, table, schema, execute insert, update scripts.

In case of File backup, a valid destination file path should be supplied from the application configuration file. It can be a shared network path with required permissions set.

```
<add key="DataBackupPath" value="\fileserver\backup" />
```

Supported backup file formats are XML, CSV. Files with the table name are created in the given data backup path. e.g.: dbo_Employee.xml The zip option is available to zip these files into one file with name "databasename.zip"

Destination

File Server
Database

Privileges associated with this feature

View_Backup – The users with this privilege can view the backup request details and status.
Create_Backup – The users with this privilege are allowed to create backup request.

Data backup service setup

The CelloSaaS package will contain DatabackupServiceSetup.msi file. Install and change the connection to point to your CelloSaaS database.

Data backup request API

```
/// <summary>
/// Creates the backup request.
/// </summary>
/// <param name="request">The request.</param>
/// <returns></returns>
/// <exception cref="System.ArgumentException">thowed when the request object contains invalid values</exception>
/// <exception cref="CelloSaaS.DataBackup.Model.BackupRequestException">thowed when any database related error occured.</exception>
string CreateBackupRequest(BackupRequest request);

/// <summary>
/// Cancels the backup request.
/// </summary>
/// <param name="requestId">The request id.</param>
/// <returns></returns>
/// <exception cref="System.ArgumentNullException">if the requestId is null or empty</exception>
/// <exception cref="CelloSaaS.DataBackup.Model.BackupRequestException">thowed when any database related error occured.</exception>
bool CancelBackupRequest(string requestId);

/// <summary>
/// Gets the backup request details.
/// </summary>
/// <param name="requestId">The request id.</param>
/// <returns></returns>
/// <exception cref="System.ArgumentException">if the requestId is null or empty</exception>
/// <exception cref="CelloSaaS.DataBackup.Model.BackupRequestException">thowed when any database related error occured.</exception>
BackupRequest GetBackupRequestDetails(string requestId);

/// <summary>
/// Searches the backup request with the given search condition.
/// </summary>
/// <param name="condition">The condition.</param>
/// <returns></returns>
/// <exception cref="System.ArgumentNullException">if the search condition object is null.</exception>
/// <exception cref="CelloSaaS.DataBackup.Model.BackupRequestException">thowed when any database related error occured.</exception>
BackupRequestSearchResult SearchBackupRequest(BackupRequestSearchCondition condition);
```

Note

1. If backup data scope is selected as Child tenants/Immediate child tenants/Selective tenants then those tenants data should also reside in the parent tenant's database.
2. Currently the source connection string for backup operation is taken from web.config connection string configuration section. Application developers are requested to change the source connection as per their need.
3. File backup generated may not be directly down-loadable by the tenant. Application developer are requested to provided such functionality if required.
4. The framework will not provide any data export functionality

22 APPENDIX

How to Guides:

These simplified How To guides will help developers to quickly follow and build applications following the instructions given.

Name of the Topic	How To Guide	Sample Application
How To - Package Management	http://techcello.com/downloads/how-to/how_to_package_management.pdf	http://techcello.com/downloads/samples/employee_management_package_management.zip
How To - Security Management	http://techcello.com/downloads/how-to/how_to_security_management.pdf	http://techcello.com/downloads/samples/employee_management_security.zip
How To - Customize Logo Theme	http://techcello.com/downloads/how-to/how_to_customize_logo_theme.pdf	http://techcello.com/downloads/samples/employee_management_logo_theme_customization.zip
How To - Customization of Basic Fields	http://techcello.com/downloads/how-to/how_to_customization_of_basic_fields.pdf	http://techcello.com/downloads/samples/employee_management_base_fields_customization.zip
How To - Extension Fields	http://techcello.com/downloads/how-to/how_to_extension_fields.pdf	
How To - Customize Entity validation	http://techcello.com/downloads/how-to/how_to_customize_entity_validation.pdf	http://techcello.com/downloads/samples/employee_management_entity_validation.zip
How To - Notification Management	http://techcello.com/downloads/how-to/how_to_notification_management.pdf	http://techcello.com/downloads/samples/employee_management_notification.zip
How To - Event Audit	http://techcello.com/downloads/how-to/how_to_event_audit.pdf	http://techcello.com/downloads/samples/employee_management_event.zip
How To - Business Rules Introduction	http://techcello.com/downloads/how-to/how_to_business_rules_introduction.pdf	http://techcello.com/downloads/samples/employee_management_business_rules.zip
How To - Entity Based Business Rules	http://techcello.com/downloads/how-to/how_to_entity_based_business_rules.pdf	http://techcello.com/downloads/samples/employee_management_business_rules.zip
How To - Open Business Rules	http://techcello.com/downloads/how-to/how_to_open_business_rules.pdf	http://techcello.com/downloads/samples/employee_management_business_rules.zip
How To - XML Rule	http://techcello.com/downloads/how-to/how_to_xml_rule.pdf	http://techcello.com/downloads/samples/employee_management_business_rules.zip
How to – Configure App Fabric Cache with Cello	http://techcello.com/downloads/how-to/how_to_appfabric_cache.pdf	
How to – Configure separate databases for each tenant	http://techcello.com/downloads/how-to/how_to_application_databases.pdf	
How To - Add custom settings	http://techcello.com/downloads/how-to/how_to_add_custom_settings.pdf	
How To – Use Entity Framework for Data Access Layer	http://techcello.com/downloads/how-to/how_to_entity_framework_data_acces_layer.pdf	http://techcello.com/downloads/samples/employee_management_using_entity_framework.zip
How To – Use ADO.NET for Data Access Layer	http://techcello.com/downloads/how-to/how_to_ado.net_data_access_layer.pdf	http://techcello.com/downloads/samples/employee_management_using_ADO.Net.zip
How To - Add a new	http://techcello.com/downloads/how-	

feature as a part of package?	to/how_to_feature_management.pdf	
How To - Implement Bulk Email notification in Cello	http://techcello.com/downloads/how-to/how_to_bulk_email_notifications.pdf	http://techcello.com/downloads/samples/employee_management_notification.zip
How To - Implement Bulk FTP notification in Cello	http://techcello.com/downloads/how-to/how_to_bulk_ftp_notifications.pdf	
How To - Implement Bulk System notification in Cello	http://techcello.com/downloads/how-to/how_to_bulk_system_notifications.pdf	

23 CONTACT INFORMATION

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

Email: support@techcello.com

Phone: +1(609)503-7163

Skype: techcello

Timings: 8.30 AM – 6.30 AM IST [Monday- Friday]