Cello How-To Guide

Business Events

Contents

1 CelloSaaS Event System 3
1.1 Event Types 3
1.2 Create Event 3
1.3 How to raise the events in the system 5
1.3.1 InProc Environment 6
1.3.2 Web Service Enviroment 6
1.4 Mapping Content Template to an Event 7
1.5 To Create and Register Job 7
1.6 To create, delete, fetch the Job details 8
1.7 Mapping Jobs to Event 8
1.8 Event and Job Audit APIs 8
1.9 Event Workflow Job Handler 9
1.10 To map Workflow Creation Job to an event 9
1.11 Mapping Workflow Job to an Event 10
1.12 Passing Workflow input 10
1.13 Create Workflow input finder class 10
1.14 Introduction to Event Audit 11
1.15 To fetch Event Audit Details via code 12
Contact Information 14

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.

Events can be managed through User Interface via Product Admin login which is a onetime process.

1.1 Event Types

CelloSaaS has two types of events.

1) Automatic CURD events

   These events will be raised automatically when the entity is created, updated and deleted.

   Need to follow the below naming convention for this event. }}

   - Create Event – {{entity Id}}_Created Ex: Employee_Created
   - Update Event – {{entity Id}}_Updated Ex: Employee_Updated
   - Delete Event – {{entity Id}}_Deleted Ex: Employee_Deleted

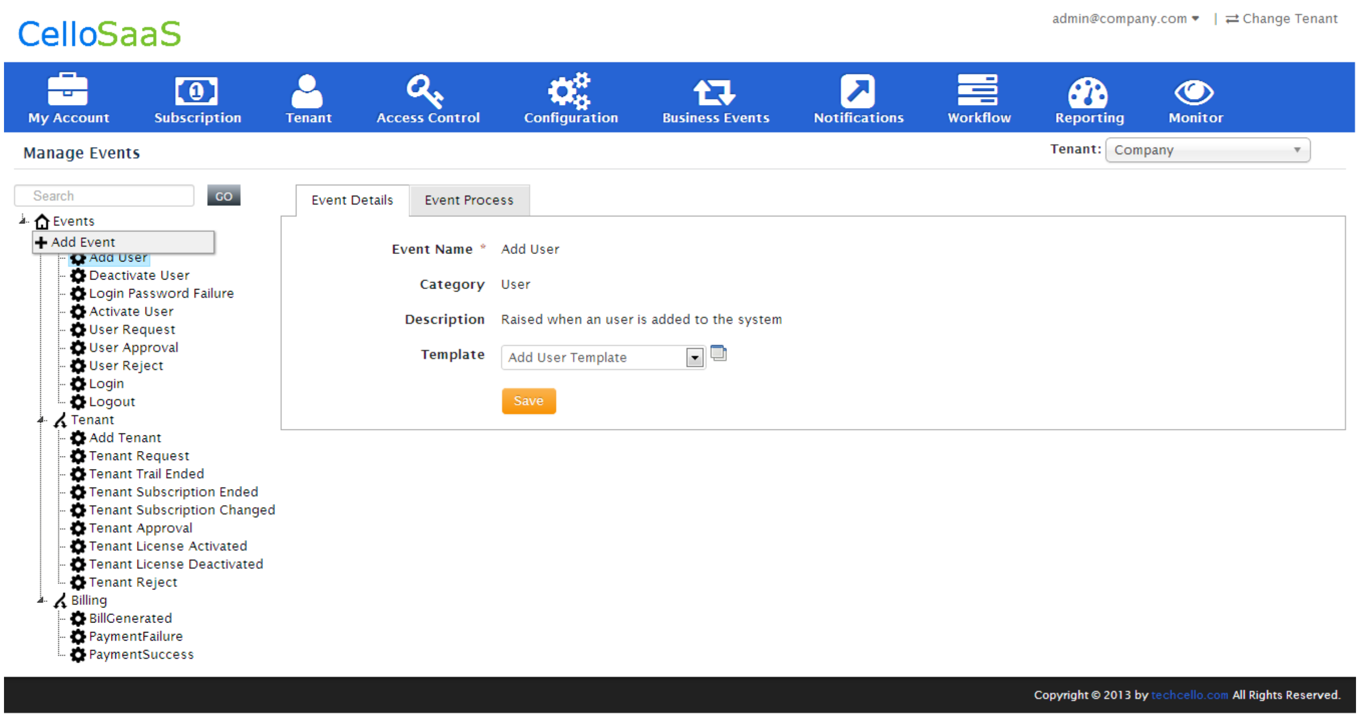
2) Custom events

   Developer need to raise the events based on their business need.

1.2 Create Event

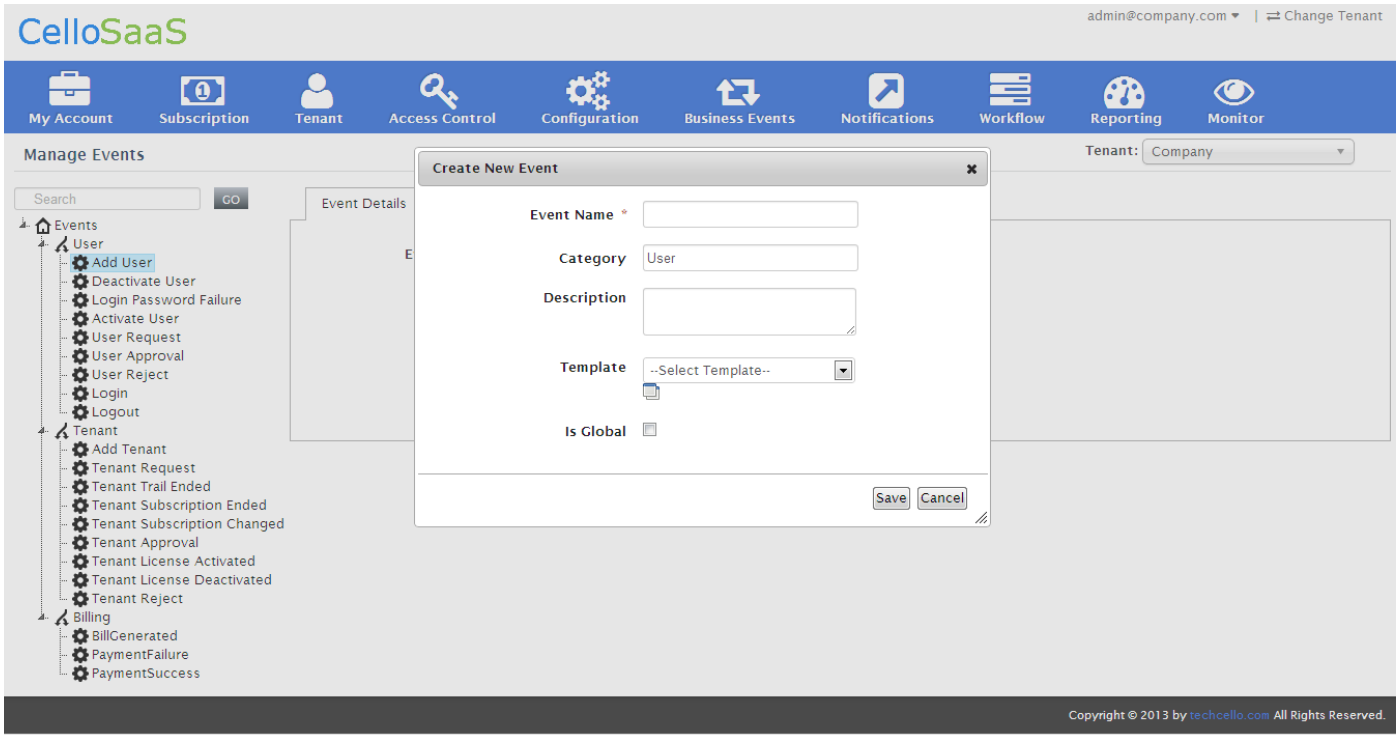
The Event can be created as follows. Go to **Mange Events -> Manage Events**

To create new event, right click the default icon.



Event Name *	Add User
Category	User
Description	Raised when an user is added to the system
Template	Add User Template

After clicking the Add Event menu the following screen will appear.



Event Name *	
Category	User
Description	
Template	--Select Template--
Is Global	☐

In the above screen, the selection of template is optional. If the template is required, then create the template.

Tip: Since the events are created and are constant, the event ids can be maintained within the code so that they can be accessed uniformly throughout the application in a consistent fashion.

```
public class EventConstants
{
   public const string EmployeeAditionEvent = "FD9E31D8-EA53-40DE-9453-147DF9BB5390";
[or]
  public const string EmployeeAdditionEvent = "Employee Addition";
}
```

1.3 How to raise the events in the system

The following sample demonstrates about how to raise an event when any employee is created. Uses the event Id or event name while register the event.

Object and XML Mode

Cello event system supports, both the entity placeholders and xml placeholders. Entity placeholders are used in inproc mode and xml placeholders is used in WCF mode.

Mapping Template

Details of mapping via UI

In XML Mode, the palceholders used needs to be as below

1. SubjectXmlValue
2. ContextXmlValue
3. TargetXmlValue

```
public void AddEmployee(Employee entity)
{
   // your code to create an employee
   // once successful then raise the event
   EventRegisterProxy.RegisterEvent(new Event
               {
                  EventId = EventConstants.EmployeeAditionEvent,
                  TenantId = UserIdentity.TenantID,
                  UserId = UserIdentity.UserId,
                  SubjectType = "Employee",
                  SubjectXmlValue = entity.SerializeToXml()
               });
}
```

In object mode, the placeholders used needs to be as below

1. SubjectValue
2. ContextValue
3. TargetValue

```
public string AddEmployeeDetails(EmployeeDetails employeeDetails)
{
        /* Write the Logic to adding employee details*/
        //To log the event
        EventRegisterProxy.RegisterEvent(new Event
        {
                EventId = EventConstant.AddEmployeeEventId,
                TenantId = UserIdentity.TenantID,
                UserId = userDetailsId,
                SubjectType = "EmployeeDetails",
                SubjectId = employeeDetails.Identifier,
                SubjectValue = employeeDetails
        });
        return employeeDetails.Identifier;
}
```

These placeholder values will be used in template processing. It helps the template engine to replace the placeholder with the corresponding values.

When the above code is executed, the CelloSaaS Event System will run the jobs configured by the Tenant against this particular event based on the targeted schedule.

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. The processing of the new events, number of events [Example: 5000], frequency etc] will be read from the configuration file. All the events / jobs are audited by CelloSaaS and are made available from the User Interface.

1.3.1 InProc Environment

This environment is referring to the in process service calls that are being made from the application. In this mode or environment, the data that can be used are C# objects or they can be serialized in XML format and then persisted as event data.

SubjectValue / ContextValue / TargetValue can contain any C# object.

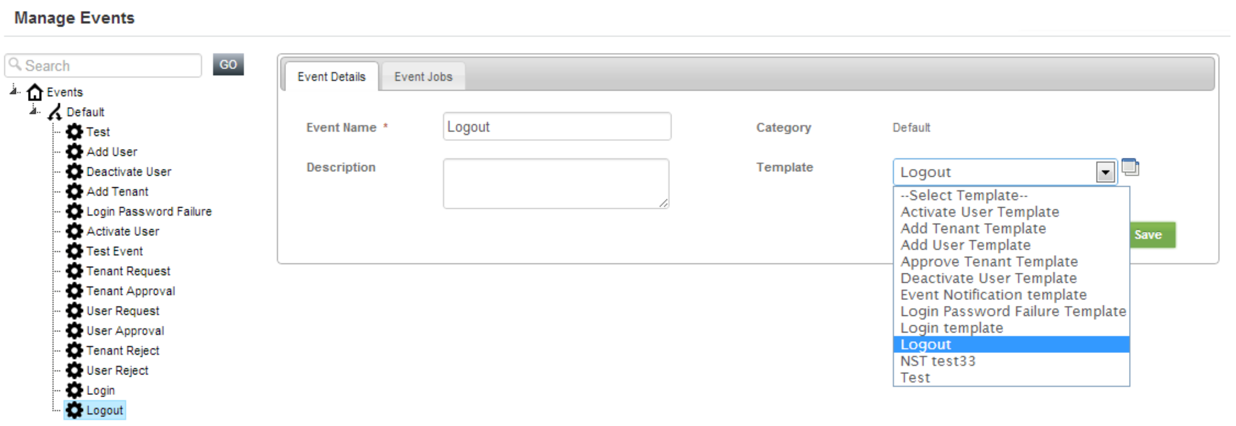
1.3.2 Web Service Enviroment

SubjectXmlValue / ContextXmlValue / TargetXmlValue will contain the serialized data of the business object as the object data is represented as XML format in web services. Persistence of the data is easy and the end user needs to use XPath expression to replace the placeholders in the template content. These data can be used in both WCF and InProc environment.

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

The events can 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.

1.4 Mapping Content Template to an Event



[Mapping Content Template to an Event]

1.5 To Create and Register Job

To Create custom Jobs follow the below steps.
Create a class implementing from CelloSaaS.EventScheduler.EventPublishingEngine.IJob interface

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

Create job details in DB using following script or ISchedulerService methods..

```
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]

1.6 To create, delete, fetch the Job details

```
interface CelloSaaS.EventScheduler.ServiceContracts.ISchedulerService
{
/// This service will add a new job record
string AddJob(Job job);

/// This service will delete the given job id
void DeleteJob(string jobId);

/// This method fetches the job details for the given job id
Job GetJob(string jobId);

}
```

1.7 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
{
        /// Adds the given job to the event.
        void AddEventJob(string eventId, string jobId, string tenantId);

        /// Deletes the job from the event.
        void DeleteEventJob(string eventId, string jobId, string tenantId);

        /// This method will search for the jobs mapped to the given event.
        Dictionary<string, Job> GetJobs(string eventId, string tenantId);
}
```

1.8 Event and Job Audit APIs

```
interface CelloSaaS.EventScheduler.ServiceContracts.IEventAuditService
{
        /// Get ActivityEventLog  details by eventId,tenantid
        Dictionary<string, Event> GetEventAudits(string eventId, string tenantId);

        /// Get all ActivityEventLog details for the tenantId.
        Dictionary<string, Event> GetEventAuditsByUserId(string userId);

        /// Searches the event audit details.
        EventAuditSearchResult GetEventAuditDetails(EventAuditSearchCondition
        eventAuditSearchCondition);
}

interface CelloSaaS.EventScheduler.ServiceContracts.IJobAuditService
{
        /// Searches the job audits with the given conditions.
        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.

1.9 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, whenever the event occur it creates the given Workflow Instance and starts the workflow.

1.10 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”;
```

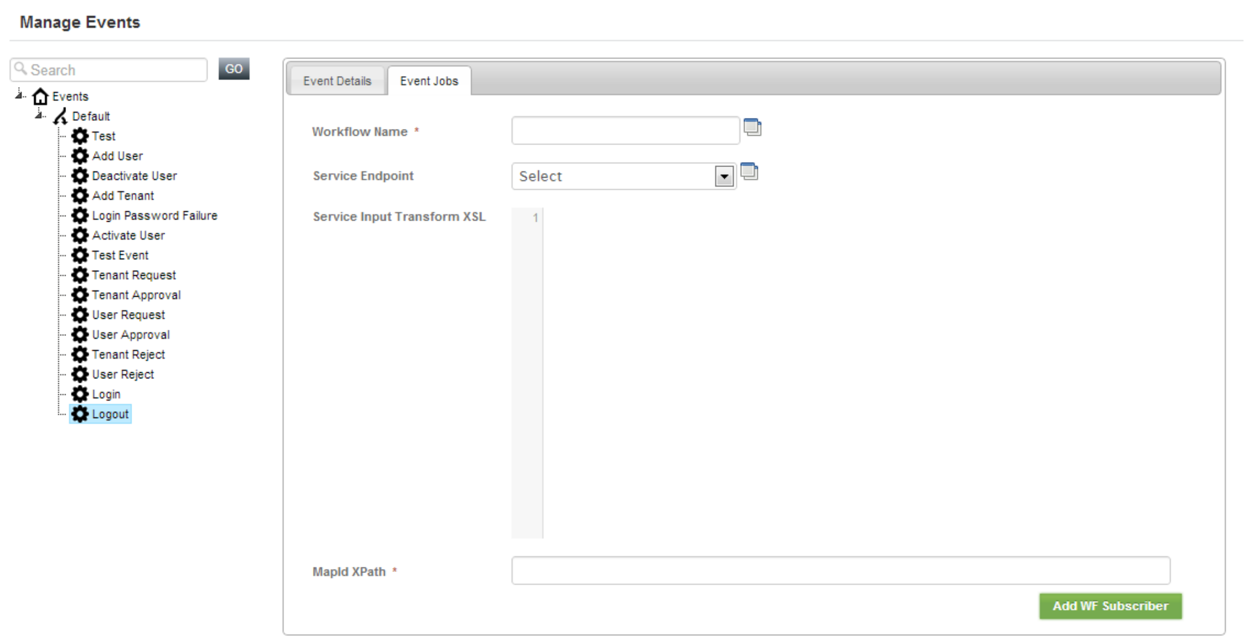
Note: MapIdXpath will be obtained from the Event Xml. So mapId need to be passed when raising the event. eg. use SubjectId or ContextId or TargetId, etc to store the mapId.

```
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.

1.11 Mapping Workflow Job to an Event



[Mapping Workflow Job to an Event.]

1.12 Passing Workflow input

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

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

1.14 Introduction to Event Audit

Event Audit can be logged for every event in cello as well as at the application level.

Let’s take the employee management system, the following events can be logged

1. When an employee/user was created.
2. When the user log in or logout from the system.
3. When the user’s password change in the system.
4. When the user is activated/deactivated.
5. When a tenant is created or approved.

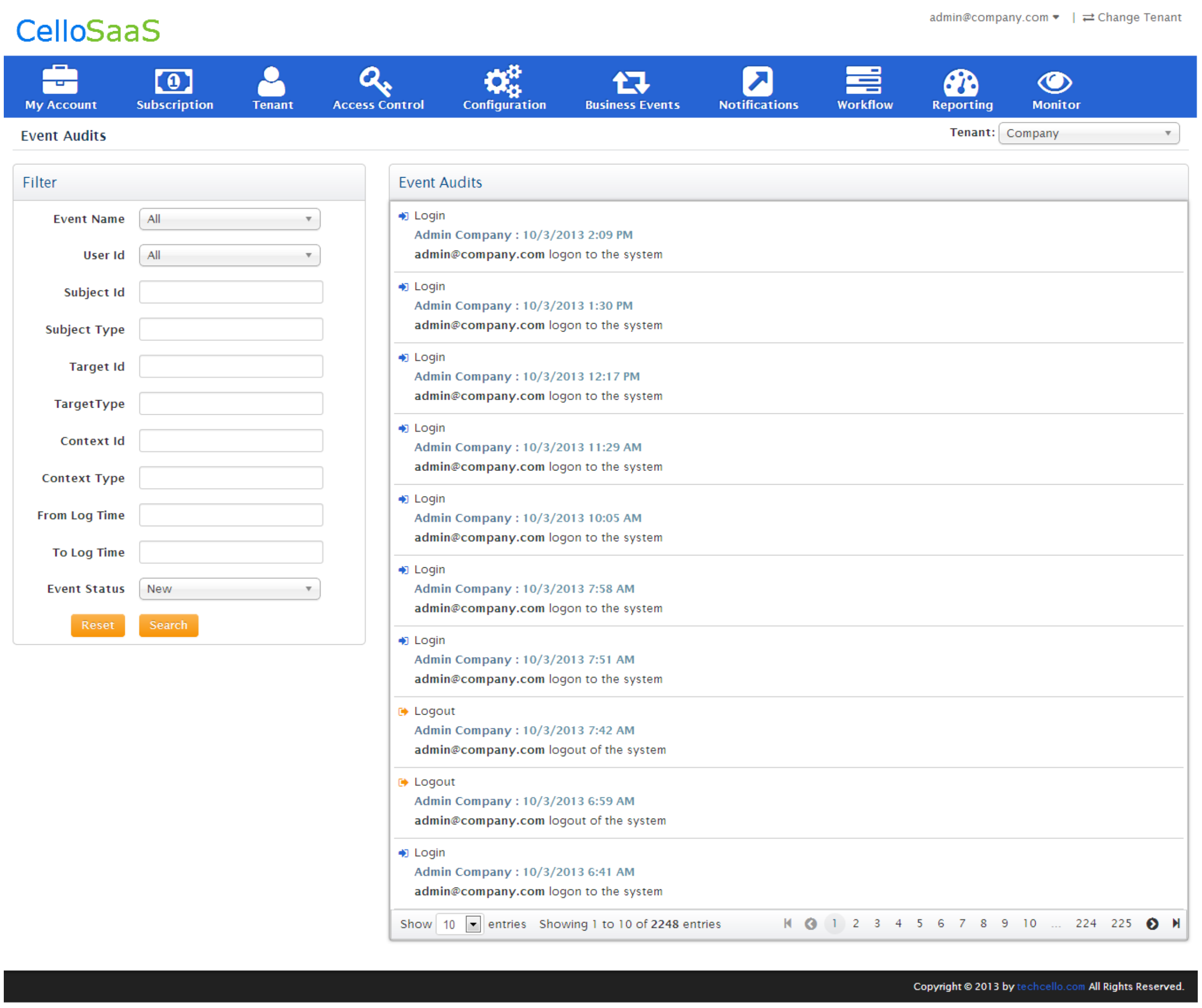
To log the event, the following step needs to be followed.

1. Create required Event through UI
2. Create Template for that event, if it requires
3. Call the Event Register API to log the event.

Use **Monitor-> Events** navigation to view all raised events.

Viewing the logged events via UI

The logged event audit can be viewed as follows.



Event Audits

Tenant: Company

Filter

Event Name	All
User Id	All
Subject Id	
Subject Type	
Target Id	
TargetType	
Context Id	
Context Type	
From Log Time	
To Log Time	
Event Status	New

Event Audits

Login
Admin Company : 10/3/2013 2:09 PM
admin@company.com logon to the system

Login
Admin Company : 10/3/2013 1:30 PM
admin@company.com logon to the system

Login
Admin Company : 10/3/2013 12:17 PM
admin@company.com logon to the system

Login
Admin Company : 10/3/2013 11:29 AM
admin@company.com logon to the system

Login
Admin Company : 10/3/2013 10:05 AM
admin@company.com logon to the system

Login
Admin Company : 10/3/2013 7:58 AM
admin@company.com logon to the system

Login
Admin Company : 10/3/2013 7:51 AM
admin@company.com logon to the system

Logout
Admin Company : 10/3/2013 7:42 AM
admin@company.com logout of the system

Logout
Admin Company : 10/3/2013 6:59 AM
admin@company.com logout of the system

Login
Admin Company : 10/3/2013 6:41 AM
admin@company.com logon to the system

Show 10 entries Showing 1 to 10 of 2248 entries 1 2 3 4 5 6 7 8 9 10 ... 224 225

You can search the event audit based on the following filters. Event Name, User Name(user Id), Subject Id, Subject Type, Target Id, Target Type, Context Id, Context Type, From Log Date, To Log Date and Event Status.

1.15 To fetch Event Audit Details via code

Namespace : CelloSaaS.EventScheduler.ServiceProxies
Class : EventAuditServiceProxy
Method: public static EventAuditSearchResult GetEventAuditDetails(EventAuditSearchCondition eventAuditSearchCondition).

- Here, EventName is the name of the event and the UserId is the logged in user's UserId.
- SubjectId/ContextId/TargetId, SubjectType/TargetType/ContextType contains user friendly string to facilitate easy searching by the end user in UI.

- FromLogTime / ToLogTime represent the event logged time. By using this, users can get the events audits between the given time frames.

Sample Code

```
public  EventAuditSearchResult GetEventAudit(EventAuditSearchCondition eventAuditsearchcondition, int?
page, string sortString, string sortDirection, int pageSize = 10)
{

        /* Write logic for set the audit search condition */

        //To get the Event audit details
        eventAuditSearchResult  = EventAuditServiceProxy.GetEventAuditDetails(eventAuditsearchcondition);

       return eventAuditSearchResult;
}
```

Contact Information

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

Email: support@techcello.com

Phone: +1(609)503-7163

Skype: techcello