Cello How-To Guide

Pre & Post Processors

techcello

Contents

1. Pre & Post Processors

CelloSaaS presents a loosely coupled, modularized, component based architecture, which allows the developers to easily extend and customize the application as per the product requirement. Cello provides end points through which developers can plug-in their custom code where ever required.

Below are the end points which help to achieve the same.

They are

1. Pre-Processor [Before CRUD Operation]
2. Post-Processor[After CRUD Operation]

Use Case: Pre-Processor

- To define actions that needs to be performed before any CRUD operation for an Entity.
- If you have any business logic validation before executing the CURD operation for the above entities then you can use Pre-Processors.

Available Entity

Pre Processor applicable for the below entities, they are

- Package
- Tenant
- User
- Role

How to define Pre-Processor

Step 1: Define Pre-Processor

- In order to define a Preprocessor, you have to first create a class file and implement ***CelloSaaS.Services.IPreProcessorProvider*** *interface*.
- Available methods in Pre-Processor interface

Methods	**Description**	**Available Entities**
PreProcessorInsert	Used to do some action for the entity data before the Insert operation	Package, Tenant, User and Role
PreProcessorUpdate	used to do some action for the entity data before	Package, Tenant,

	the Update operation	User and Role
PreProcessorDelete	used to do some action for the entity data before the Delete/Deactivate operation	Package, Tenant, User and Role
PreProcessorPermanent Delete	used to do some action for the entity data before the Permanent Delete operation	Package, Tenant, User and Role
PreProcessorActivate	used to do some action the entity data before the Activate operation	User

Step 2: Register Pre-Processor

- Register the Pre-Processors in the ***web.config*** file inside the <processors> section.
- Use the following pre-processors name(Naming Convention) while doing the configuration.

Pre-Processor Name	**Entity**
PackagePreProcessor	Package
TenantPreProcessor	Tenant
UserDetailsPreProcessor	User
RolePreProcessor	Role

Sample Pre-Processor class

```
namespace WebApplication.Models
{
    /// <summary>
    /// Tenant Validation Pre Processor to handle the pre-processor
    /// </summary>
    public class TenantValidationPreProcessor : IPreProcessorProvider
    {
        /// <summary>
        /// This PreProcessor can be used for validating the tenant data during the Insert
        operation
        /// </summary>
        /// <param name="entity">Entity Object</param>
        /// <param name="args">Object Params</param>
        /// <returns>
        /// Success - True / False
        /// </returns>
        public bool PreProcessorInsert(object entity, params object[] args)
        {
             // Validate the Entity Parameters here and return the status accordingly
             throw new NotImplementedException();
        }
```

```
/// <summary>
/// This PreProcessor can be used for validating the tenant data during the update
operation
/// </summary>
/// <param name="referenceId">
/// Reference Id
/// <remarks> This will be identfier of the tenant that is undergoing update </remarks>
/// </param>
/// <param name="entity">Entity Object</param>
/// <param name="args">Object Params</param>
/// <returns>
/// Success - True / False
/// </returns>
public bool PreProcessorUpdate(string referenceId, object entity, params object[] args)
{
     // Process the entity for the update operation here and return the status
     throw new NotImplementedException();
}

/// <summary>
/// This PreProcessor can be used for validating the tenant data during the soft delete
operation
/// </summary>
/// <param name="referenceId">Reference Id</param>
/// <remarks> This will be identfier of the tenant that is undergoing update </remarks>
/// <param name="entity">Entity Object</param>
/// <param name="args">Object Params</param>
/// <returns>
/// Success - True / False
/// </returns>
public bool PreProcessorDelete(string referenceId, object entity, params object[] args)
{
     // handle the soft delete pre processor here
     throw new NotImplementedException();
}

/// <summary>
/// This PreProcessor can be used for validating the tenant data during the permanent
delete operation
/// </summary>
/// <param name="referenceId">Reference Id</param>
/// <remarks> This will be identfier of the tenant that is undergoing update </remarks>
/// <param name="entity">Entity Object</param>
/// <param name="args">Object Params</param>
/// <returns>
/// Success - True / False
/// </returns>
public bool PreProcessorPermanentDelete(string referenceId, object entity, params
object[] args)
{
     // Handle the permanent deletion pre processing here
     throw new NotImplementedException();
}
```

```
        /// <summary>
        /// This PreProcessor can be used for validating the tenant data during the activate
        operation
        /// </summary>
        /// <param name="referenceId">Reference Id</param>
        /// <remarks> This will be identfier of the tenant that is undergoing update </remarks>
        /// <param name="entity">Entity Object</param>
        /// <param name="args">Object Params</param>
        /// <returns>
        /// Success - True / False
        /// </returns>
        public bool PreProcessorActivate(string referenceId, object entity, params object[] args)
        {
             // Handle the activation pre processing here
             throw new NotImplementedException();
        }
    }
}
```

Pre-processor Configuration

```
<processors>
      <preprocessors>
            <add name="<Pre-ProcessorName>" assembly="<AssemblyName>"
            type="<ClassTypeName>"/>
      </preprocessors>
</processors>
```

Pre-ProcessorName – Use one of the pre-processor name based on the entity.

AssemblyName – Assembly name of the implemented class

Type – Fully qualified name of the implemented class

Sample Configuration:

```
<processors>
      <preprocessors>
            <add name="TenantPreProcessor" assembly="WebApplication"type="WebApplication
            .Models.TenantValidationPreProcessor"/>
      </preprocessors>
</processors>
```

Post-Processor

Use Case: Pre-Processor

- This is used to define actions that need to be performed after any CRUD operation.
- If you have any business logic to be implemented after executing the CURD operation for the above entities then you can use Pre-Processors.

Step 1: Define Post-Processor

- In order to define a Preprocessor, you have to first create a class file and inhert form interface called ***IPostProcessorProvider***, which is available in ***CelloSaaS.Services***
- Available methods in Post-Processor interface

Methods	**Description**	**Available Entities**
PostProcessorInsert	Used to do some action the entity data after the Insert operation	Package, Tenant, User and Role
PostProcessorUpdate	used to do some action the entity data after the Update operation	Package, Tenant, User and Role
PostProcessorDelete	used to do some action the entity data after the Delete/Deactivate operation	Package, Tenant, User and Role
PostProcessorPermanentDelete	used to do some action the entity data after the Permanent Delete operation	Package, Tenant, User and Role
PostProcessorActivate	used to do some action the entity data after the Activate operation	User

Step 2: Register Post-Processor

- Register the Post-Processors in the web.config file inside the <processors> section.
- Use the following pre-processors name while doing the configuration.

Pre-Processor Name	**Entity**
PackagePostProcessor	Package

TenantPostProcessor	Tenant
UserDetailsPostProcessor	User
RolePostProcessor	Role

Sample Pre-Processor class

```
namespace WebApplication.Models
{
   /// <summary>
   /// Tenant Validation Pre Processor to handle the pre-processor
   /// </summary>
   public class TenantValidationPostProcessor : IPostProcessorProvider
   {
      /// <summary>
      /// This PostProcessor can be used for validating the tenant data during the Activate operation
      /// <param name="referenceId">The reference identifier.</param>
      /// <param name="entity">Entity Object</param>
      /// <param name="args">Object Params</param>
      /// <returns>
      /// Success - True / False
     public bool PostProcessorActivate(string referenceId, object entity, params object[] args)
      {
          // Handle the activation post processing here
          throw new NotImplementedException();
      }

      /// <summary>
      /// This PostProcessor can be used for validating the tenant data during the soft delete operation
      /// <param name="referenceId">Reference Id</param>
      /// <remarks> This will be identfier of the tenant that is undergoing update </remarks>
      /// <param name="entity">Entity Object</param>
      /// <param name="args">Object Params</param>
      /// Success - True / False
      /// </returns>
      public bool PostProcessorDelete(string referenceId, object entity, params object[] args)
      {
          // handle the soft delete post processor here
          throw new NotImplementedException();
      }
      /// <summary>
      /// This PostProcessor can be used for validating the tenant data during the Insert operation
      /// <param name="entity">Entity Object</param>
      /// <param name="args">Object Params</param>
      /// Success - True / False
      public bool PostProcessorInsert(string referenceId, object entity, params object[] args)
      {
          // handle the insert post processor here
          throw new NotImplementedException();
      }
```

```
/// <summary>
        /// This PostProcessor can be used for validating the tenant data during the permanent delete
        operation
      /// </summary>
      /// <param name="referenceId">Reference Id</param>
        /// <remarks> This will be identfier of the tenant that is undergoing update </remarks>
      /// <param name="entity">Entity Object</param>
      /// <param name="args">Object Params</param>
      /// <returns>
      /// Success - True / False
      /// </returns>
        public bool PostProcessorPermanentDelete(string referenceId, object entity, params object[]
        args)
      {
          // Handle the permanent deletion post processing here
          throw new NotImplementedException();
      }

      /// <summary>
        /// This PostProcessor can be used for validating the tenant data during the update operation
      /// </summary>
      /// <param name="referenceId">
      /// Reference Id
        /// <remarks> This will be identfier of the tenant that is undergoing update </remarks>
      /// </param>
      /// <param name="entity">Entity Object</param>
      /// <param name="args">Object Params</param>
      /// <returns>
      /// Success - True / False
      /// </returns>
        public bool PostProcessorUpdate(string referenceId, object entity, params object[] args)
       {
          // handle the update post processor here
          throw new NotImplementedException();
       }
    }
}
```

Post-Processor Configuration

```
<processors>
        <postprocessors>
                <add name="<Post-ProcessorName>" assembly="<AssemblyName>"
                type="<ClassTypeName>"/>
        </postprocessors>
</processors>
```

Post-Processor Name – Use one of the post-processor name based on the entity.

AssemblyName – Assembly name of the implemented class

Type – Fully qualified name of the implemented class

Sample Configuration

```
<processors>
        <postprocessors>
                <add name="TenantPostProcessor" assembly="WebApplication"type="WebApplicati
                on.Models.TenantValidationPostProcessor"/>
        </postprocessors>
</processors>
```

2. 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