Contents

DevForce defines two types of validation. Property level validation and instance level validation.

Property level validation executes automatically before and/or after a property value is changed. This validation automatically occurs on the client and results in immediate feedback to the UI. Instance level validation is performed on the server before an entity is persisted to the database. The purpose of instance level validation is to make sure that an entity and it's related entities are in a valid state in order to maintain data integrity. This approach ensures that no invalid data gets passed the server, but doesn't give the UI much control. The server simply throws exceptions. It is often desired to perform the instance level validation on the client to save a roundtrip to the server.

To learn more about how to define validation rules in DevForce visit the Validate topic.

Client-side validation in Punch

Punch automatically performs client-side validation and if a validation error occurs, a ValidationException is thrown.

Customizing validation

The out-of-the-box client-side validation is enough to get started, but eventually as a developer one wants to customize what should happen if a validation error occurs or customize the kind of validation rules that should be performed.

Punch provides a mechanism to take control over what should happen if a validation error occurs. This control is given through the IValidationErrorNotification interface. Simply implement this interface on one or more classes and Punch will instead of throwing an exception, notify your application of any validation errors via the interface, and cancel the current save operation.

The following snippet shows a common implementation, which takes the validation errors and republishes them via the EventAggregator.

```
using Cocktail;
using IdeaBlade.Validation;
namespace Common.Validation
{
  public class ValidationErrorHandler : IValidationErrorNotification
   {
     #region IValidationErrorNotification Members
     public void OnValidationError(VerifierResultCollection validationErrors)
      {
        if (validationErrors.HasErrors)
            EventFns.Publish(new ValidationErrorMessage(validationErrors));
      }
     #endregion
  }
}
```

The second customization a developer may want to do is implement custom validation rules on top of any DevForce Verifiers. We have previously introduced the EntityManagerDelegate (see EntityManager providers for more information). The EntityManagerDelegate provides a Validate method, which is automatically called for every entity that requires validation.

The following snippet shows an example implementation of the Validate method. It calls a method provided by a base class from which every entity extends.

```
public class EntityManagerDelegate : EntityManagerDelegate<TempHireEntities>
{
  public override void Validate(object entity, VerifierResultCollection validationErrors)
   {
      var entityBase = entity as EntityBase;
     if (entityBase != null)
        entityBase.Validate(validationErrors);
   }
}
```

Executing custom validation rules on the server

It is good practice, to make sure the server also knows about the custom validation and executes it before saving an entity. In order to execute the custom validation logic on the server, an *EntityServerSaveInterceptor* needs to be implemented, for example like so:

```
public class SaveInterceptor : EntityServerSaveInterceptor
{
  protected override bool ValidateSave()
   {
     base.ValidateSave();
     // Create a sandbox to do the validation in.
     var em = new EntityManager(EntityManager);
      em.CacheStateManager.RestoreCacheState(EntityManager.CacheStateManager.GetCacheState());
     // Find all entities supporting custom validation
     List<EntityBase> entities =
         em.FindEntities(EntityState.AllButDetached).OfType<EntityBase>().ToList();
     foreach (EntityBase e in entities)
      {
         EntityAspect entityAspect = EntityAspect.Wrap(e);
        if (entityAspect.EntityState.IsDeletedOrDetached()) continue;
         var validationErrors = new VerifierResultCollection();
         e.Validate(validationErrors);
         validationErrors =
           new VerifierResultCollection(entityAspect.ValidationErrors.Concat(validationErrors.Errors));
         validationErrors.Where(vr => !entityAspect.ValidationErrors.Contains(vr))
            .ForEach(entityAspect.ValidationErrors.Add);
        if (validationErrors.HasErrors)
           throw new EntityServerException(validationErrors.Select(v => v.Message).ToAggregateString("\n"),
                                           null,
                                            PersistenceOperation.Save, PersistenceFailure.Validation);
      }
     return true;
   }
}
```