Extended Creation

When a Task is created, a child Task Item record is created for each Task Form Item associated with the corresponding Task Definition.

After creation, a Task can have additional Task Items created and associated to the Task record. By using Logic Service Script Classes or by adding additional API when programmatically creating a Task, new Task Items can be added according to any definable logic.

By using Custom Properties in combination with this technique, Tasks can easily have dynamically generated Task Items based upon Job, Product, Shift, or any other context.

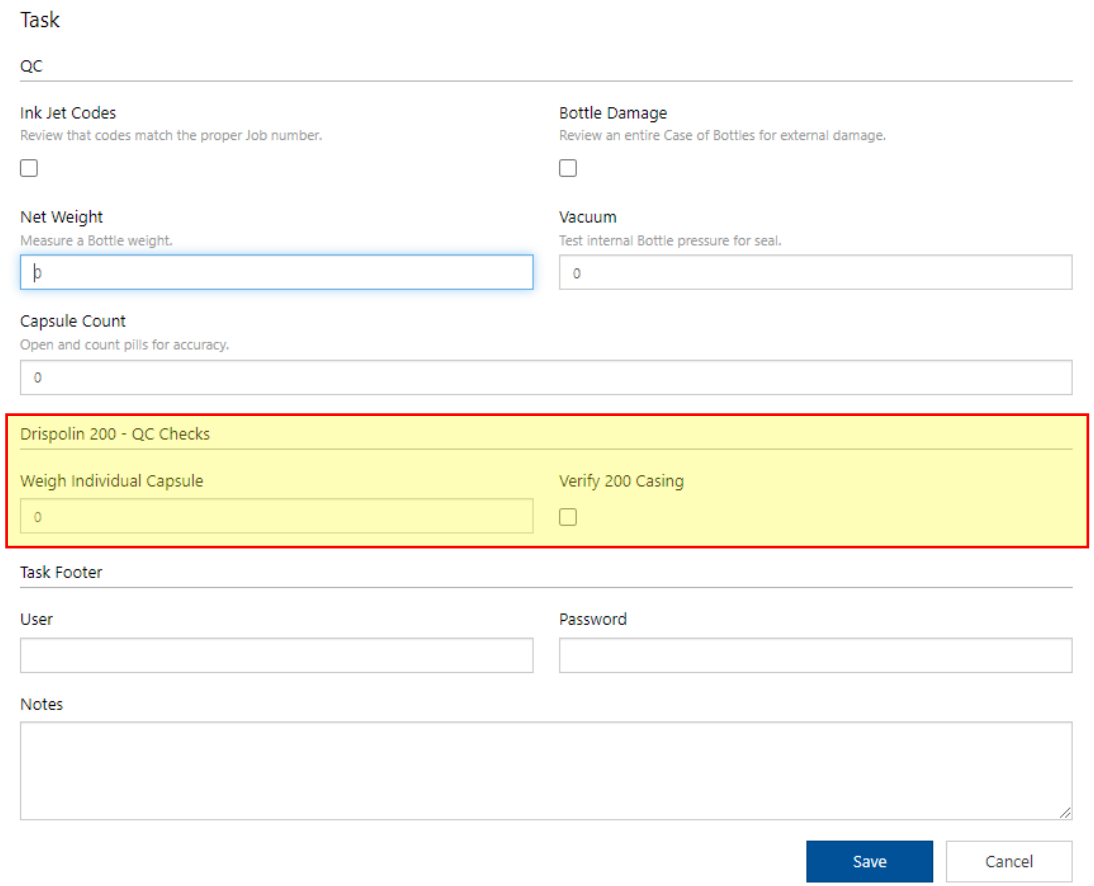
Note that this does not work when viewing a brand-new Task record, as the Task Definition script will now have had the chance to execute.

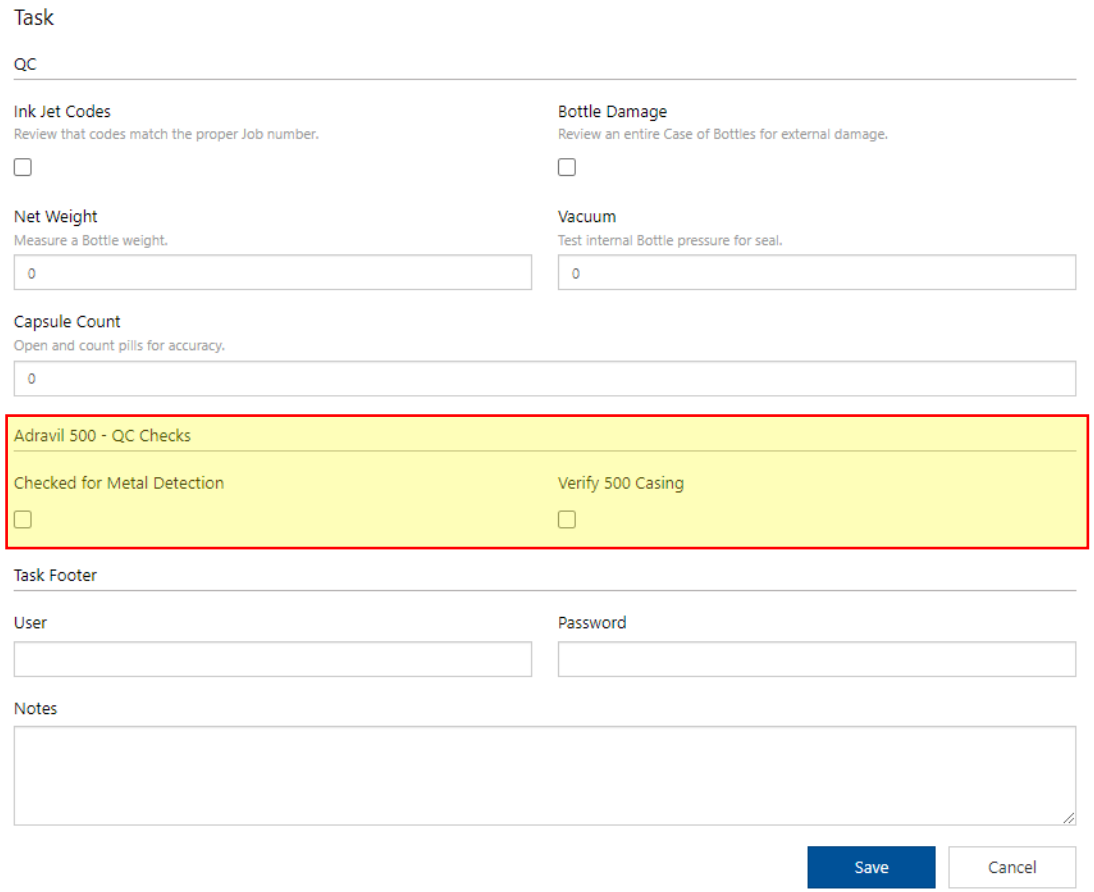
```
public class QcTaskExtension : ETS.Core.Scripting.TaskDefinitionScriptClassBase
{
  /// ************************************************************************
  public override void PostScanTaskCreated(IPostScanTaskCreatedContext context)
  {
    var task = context.Api.Data.DbTask.Load.ByID(context.TaskID);
    var product = context.Api.Data.DbProduct.Load.ByID(task.ProductID);
    if(product != null)
    {
      var qcGroupKey = product.CustomProperties.GetAsString("ProductTasking.QC.Key", "");
      var qcGroup = context.Api.Data.DbTaskFormItemGroup.Load.ByKey(qcGroupKey);
      if(qcGroup != null)
      {
        var tfiList = context.Api.Data.DbTaskFormItem.GetList.ForTaskFormItemGroupID(qcGroup.ID);
        var uow = context.Api.CreateUnitOfWork();
        foreach(var tfi in tfiList)
        {
          var ti = context.Api.Data.DbTaskItem.Create.FromParentTaskAndTaskFormItem(task, tfi);
          ti.Value = tfi.DefaultValue;
          context.Api.Data.DbTaskItem.Save.InsertAsNew(ti, uow).ThrowIfFailed();
        }
        var result = uow.ExecuteReturnsResultObject();
        if(!result.Success)
        {
          context.Api.Util.LogCustom.WriteErrorsFromResultObject(result, "QC Task Extension");
        }
      }
    }
  }
}
```

Tasks

Extended Creation

The example below depicts a sample Task Definition and various Task iterations based upon the Product associated to the Task.



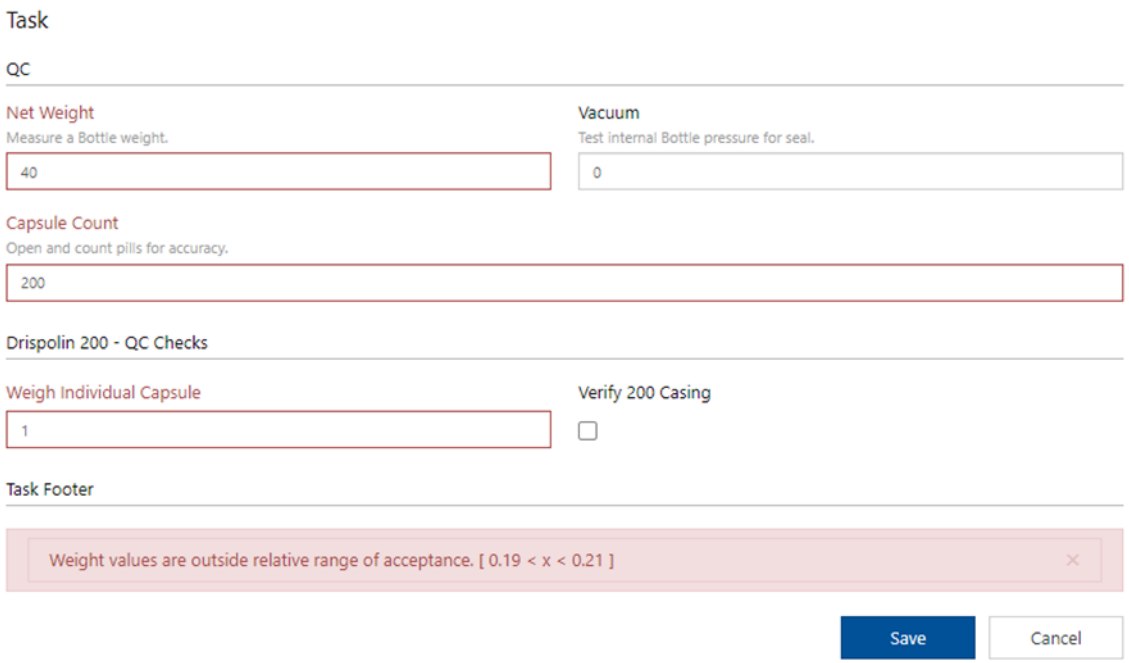


Tasks

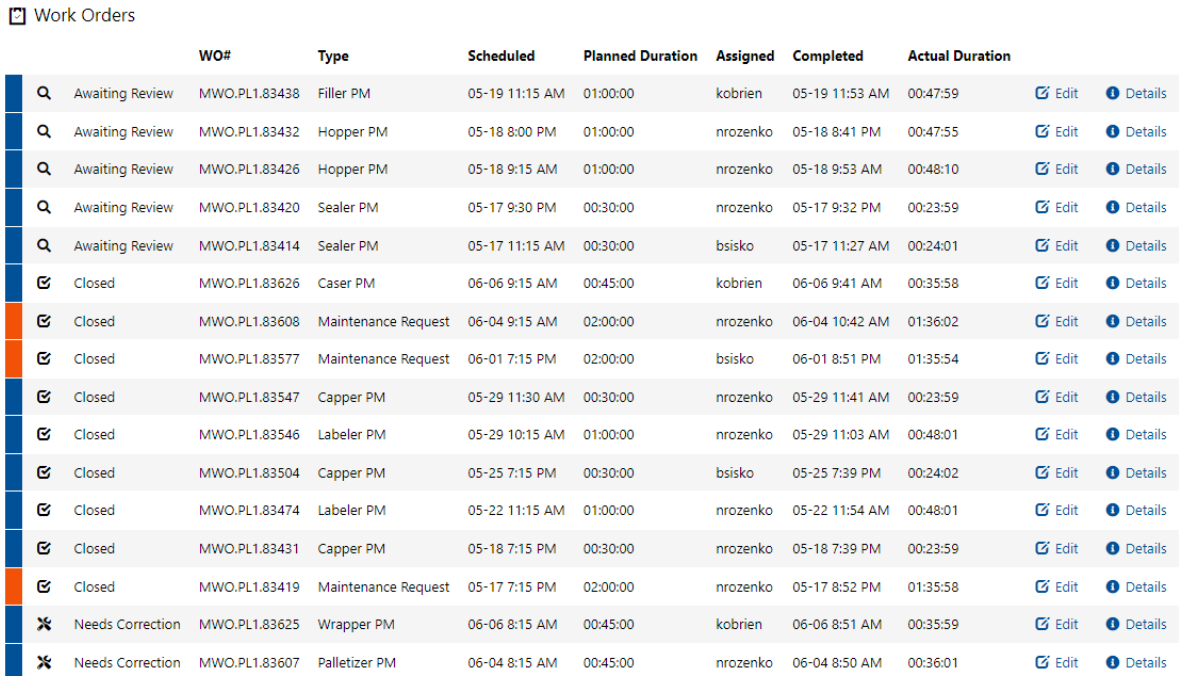
Extended Processing

While Task Form Items have basic validation options, Tasks can have additional validation applied to them.
Alternatively, Tasks can have post-processing applied using Logic Service Script Classes to define either validate or finalize a Task.

In this first example, the Task has validation applied on Web Page as part of the **Save_Click** method. It uses multiple input values for the validation. Validation errors can be shown as Errors or Warnings, with feedback displayed immediately on the page.



In this next example, the Task has processing applied after being saved as part of a Task Definition Script Class. The result is being saved as part of the Task record, as both the **PassFail** state and the User State for later visualization.



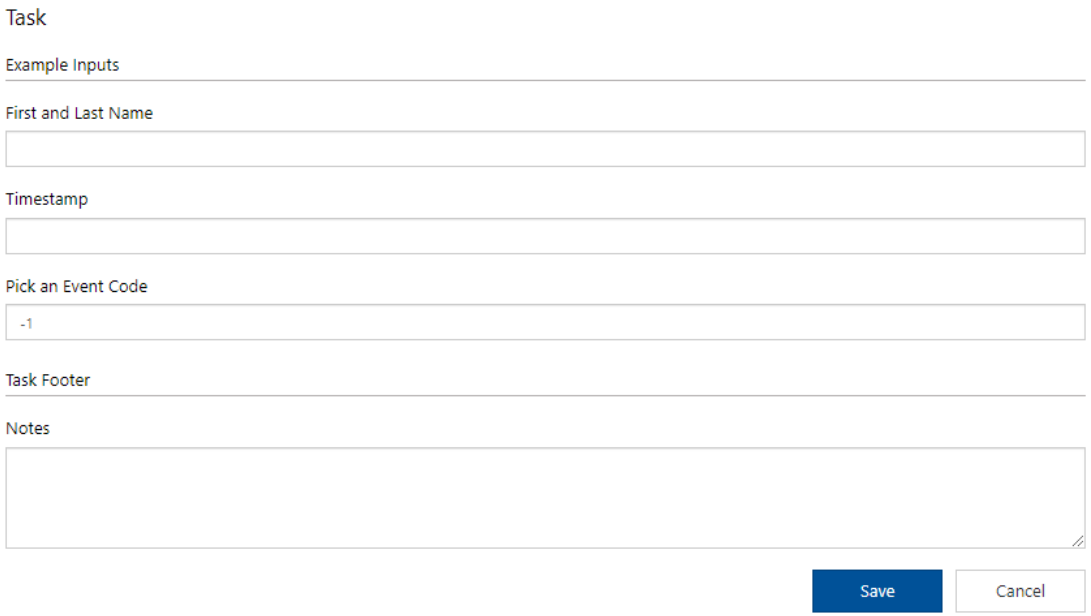
	WO#	Type	Scheduled	Planned Duration	Assigned	Completed	Actual Duration		
Awaiting Review	MWO.PL1.83438	Filler PM	05-19 11:15 AM	01:00:00	kobrien	05-19 11:53 AM	00:47:59	Edit	Details
Awaiting Review	MWO.PL1.83432	Hopper PM	05-18 8:00 PM	01:00:00	nrozenko	05-18 8:41 PM	00:47:55	Edit	Details
Awaiting Review	MWO.PL1.83426	Hopper PM	05-18 9:15 AM	01:00:00	nrozenko	05-18 9:53 AM	00:48:10	Edit	Details
Awaiting Review	MWO.PL1.83420	Sealer PM	05-17 9:30 PM	00:30:00	nrozenko	05-17 9:32 PM	00:23:59	Edit	Details
Awaiting Review	MWO.PL1.83414	Sealer PM	05-17 11:15 AM	00:30:00	bsisko	05-17 11:27 AM	00:24:01	Edit	Details
Closed	MWO.PL1.83626	Caser PM	06-06 9:15 AM	00:45:00	kobrien	06-06 9:41 AM	00:35:58	Edit	Details
Closed	MWO.PL1.83608	Maintenance Request	06-04 9:15 AM	02:00:00	nrozenko	06-04 10:42 AM	01:36:02	Edit	Details
Closed	MWO.PL1.83577	Maintenance Request	06-01 7:15 PM	02:00:00	bsisko	06-01 8:51 PM	01:35:54	Edit	Details
Closed	MWO.PL1.83547	Capper PM	05-29 11:30 AM	00:30:00	nrozenko	05-29 11:41 AM	00:23:59	Edit	Details
Closed	MWO.PL1.83546	Labeler PM	05-29 10:15 AM	01:00:00	nrozenko	05-29 11:03 AM	00:48:01	Edit	Details
Closed	MWO.PL1.83504	Capper PM	05-25 7:15 PM	00:30:00	bsisko	05-25 7:39 PM	00:24:02	Edit	Details
Closed	MWO.PL1.83474	Labeler PM	05-22 11:15 AM	01:00:00	nrozenko	05-22 11:54 AM	00:48:01	Edit	Details
Closed	MWO.PL1.83431	Capper PM	05-18 7:15 PM	00:30:00	nrozenko	05-18 7:39 PM	00:23:59	Edit	Details
Closed	MWO.PL1.83419	Maintenance Request	05-17 7:15 PM	02:00:00	nrozenko	05-17 8:52 PM	01:35:58	Edit	Details
Needs Correction	MWO.PL1.83625	Wrapper PM	06-06 8:15 AM	00:45:00	kobrien	06-06 8:51 AM	00:35:59	Edit	Details
Needs Correction	MWO.PL1.83607	Palletizer PM	06-04 8:15 AM	00:45:00	nrozenko	06-04 8:50 AM	00:36:01	Edit	Details

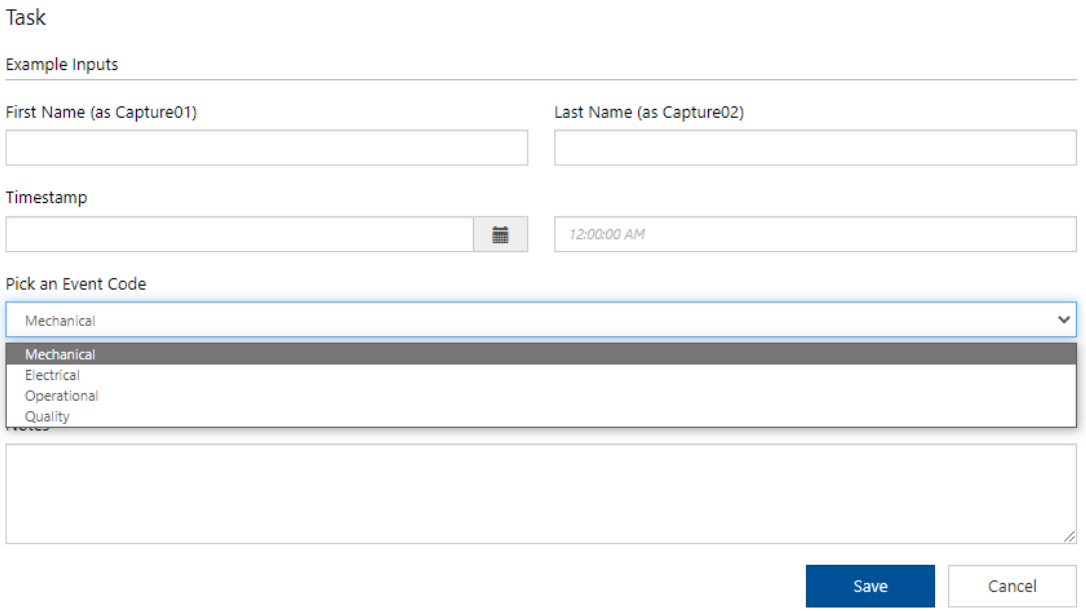
Tasks

Extended Visualization

The Task Items Content Part provide default visualization for TaskData objects on a Visual Page. Using the AddCustomTaskParts method, individual parts can be overridden to use any TrakSYS control. This allows for situational control over how a Task is displayed. In additional to changing how the Input is Visualized, additional properties and fields from the Task Item can be mapped to allow for additional data collection.

In the example below, each Input is being replaced with a more suitable part rendering. While this uses existing Content Parts, the properties that drive them have been programmatically set to allow for implementation-specific requirements to be met.





Extended Visualization

An example of the AddCustomTaskParts method is shown below. To assist with readability, it has been refactored to have each Task Item processed in individual methods based upon the associated Task Form Item's Key. An example of this can be seen in the method below it, which is being used to create the Timestamp input.

In this example, a TsInputDateTime object is being created. This matches the same Part Key of the Date Time Picker part that can be found the Parts Catalogue in a Visual Page. Unlike a Visual Page, there is no Part Property Editor. Instead, each Part Property is being set through the matching object's properties, using similar or matching property names.

Many of the Properties can be set automatically using the AddTaskItemPartInfo object, which is provided by the method as the "info" object. As seen in this example, it contains basic information, like the default CaptionEx and FormKey values. Using this info object, an input can easily be rendered using the standard configuration options. All standard TrakSYS Content Parts can be added using this pattern.

If custom visualization is not needed for an input, it can be skipped by having the AddCustomTaskParts method return false. In the example shown, the method will only return true if one of the four defined Keys are found. For any other value, the default case of the case statement will be used, which returns false and will instead result in the standard Visualization being used.

```
/// ************************************************************
protected bool AddCustomTaskParts(ETS.Ts.Core.ContentParts.Forms.TsInputTaskItems.AddTaskItemPartInfo info)
{
 switch(info.TaskFormItem.Key)
 {
   case "FirstAndLast":
     this.ProcessInfoForKey_FirstAndLast(info);
     break;
   case "DateTime_Timestamp":
     this.ProcessInfoForKey_Timestamp(info);
     break;
   case "EventCode":
     this.ProcessInfoForKey_EventCode(info);
     break;
   case "ReadySignOff":
     this.ProcessInfoForKey_ReadySignOff(info);
     break;
   default:
     return false;
 }
  return true;
}
 private void ProcessInfoForKey_Timestamp(ETS.Ts.Core.ContentParts.Forms.TsInputTaskItems.AddTaskItemPa
 {
   var part = new ETS.Ts.Core.ContentParts.Forms.TsInputDateTime();
   part.ID = info.PartID;
   part.CaptionEx = info.CaptionEx;
   part.CaptionSubEx = info.DescriptionEx;
   part.FormKey = info.FormKey;
   part.TotalWidthBs = info.TaskFormItem.WidthBs;

   part.DateTimeMode = DateTimeMode.DateTime;
   part.LayoutMode = LayoutMode.Horizontal;

   part.CssClassExtra = info.CssExtra;
   info.AddContentPart(part);
 }
```