Preventing Legacy Change Management from Adding Items in Active Change Orders

This section explains how to set up a validation method that stops you from adding items to legacy change processes when those items are already part of an active Change Order in UCM.

This method runs automatically whenever you add or update Affected Items in a legacy change process. It looks at the relationship type, collects the Change-Controlled Item (CCI) IDs from the **new_item_id** and **affected_id** fields, and then uses the **GetAffectedItemsInUcmChange** action to check whether any of those items are already part of an active Change Order.

Creating the Method

1. **Log in with Administrator privileges**
   - Sign in as a member of the **Administrators** group.
2. **Navigate to Methods**
   - Navigate to Administration → Methods.
   - Click "Create New" to open a new Method form.
3. Fill in the required properties:

Field	Description	Value
Name	Identifier for the method	ucm_NotInAnyActiveChangeOrder
Comment	Explanation of method purpose	Method validates that Affected Items are not already in any active Change Order. This method prevents adding items to legacy change management systems (Simple ECO, Express ECO, etc.) if they are already in active Change Orders.
Execution allowed to	Identity with permission to execute	Administrators
Method Type	Programming language	C#

4. Paste the method code (see below) into the code editor.

```
    innovator = this.getInnovator();
    var actionProcessor = CCO?.GetService<Aras.Mediator.IActionProcessor>()
    Item peAffectedItemRelationships = this.getRelationships($"{this.getType()} Affected Item");
    if (peAffectedItemRelationships.getItemCount() == 0)
    {

        return innovator.newResult("OK");
    }
    var validationResult = ValidateAffectedItemsNotInActiveChangeOrders(peAffectedItemRelationships,
    if (validationResult.HasConflicts)
    {
        string errorMessage = BuildConflictErrorMessage(validationResult);
        return innovator.newError(errorMessage);
    }
    return innovator.newResult("OK");
}
private Innovator innovator;
private class AffectedItemValidationResult
{
    public Dictionary<string, Dictionary<string, List<string>>> ConflictsByChangeOrderAndType { get;
    public Dictionary<string, AffectedItemMetadata> ItemMetadataByConfigId { get; set; }
    public bool HasConflicts => ConflictsByChangeOrderAndType.Count > 0;
    public AffectedItemValidationResult()
    {
        ConflictsByChangeOrderAndType = new Dictionary<string, Dictionary<string, List<string>>>();
        ItemMetadataByConfigId = new Dictionary<string, AffectedItemMetadata>();
    }
}
private class AffectedItemMetadata
{
    public string ItemName { get; set; }
    public string ItemTypeId { get; set; }
    public AffectedItemMetadata(string itemName, string itemTypeId)
    {
        ItemName = itemName ?? string.Empty;
        ItemTypeId = itemTypeId ?? string.Empty;
    }
}
private AffectedItemValidationResult ValidateAffectedItemsNotInActiveChangeOrders(
    Item peAffectedItemRelationships,
    Aras.Mediator.IActionProcessor actionProcessor)
{
    var result = new AffectedItemValidationResult();
    int relationshipCount = peAffectedItemRelationships.getItemCount();
    if (relationshipCount == 0)
    {
        return result;
    }
    var idsToCheck = ExtractChangeControlledItemIds(peAffectedItemRelationships);
    if (idsToCheck.Count == 0)
    {
        return result;
    }
    Item conflictingRelationships = FindAffectedItemsInActiveChangeOrders(idsToCheck, actionProcesso
    if (conflictingRelationships.getItemCount() == 0)
    {
```

```
        return result;
    }

    PopulateConflicts(conflictingRelationships, result);
    return result;

}
private HashSet<string> ExtractChangeControlledItemIds(Item peAffectedItemRelationships)
{
    var cciIds = new HashSet<string>();
    int relationshipCount = peAffectedItemRelationships.getItemCount();
    for (int i = 0; i < relationshipCount; i++)

    {

        Item relationship = peAffectedItemRelationships.getItemByIndex(i);
        Item affectedItem = relationship.getRelatedItem();
        if (affectedItem == null || affectedItem.isEmpty())

        {

            continue;
        }
        string newItemConfigId = ExtractConfigIdFromProperty(affectedItem, "new_item_id");

        string affectedItemConfigId = ExtractConfigIdFromProperty(affectedItem, "affected_id");

        if (!string.IsNullOrEmpty(newItemConfigId))

        {

            cciIds.Add(newItemConfigId);

        }

        if (!string.IsNullOrEmpty(affectedItemConfigId))

        {

            cciIds.Add(affectedItemConfigId);

        }

    }
    return cciIds;
}
private Item FindAffectedItemsInActiveChangeOrders(

    HashSet<string> idsToCheck,
    Aras.Mediator.IActionProcessor actionProcessor)
{
    Item activeChangeOrderFilter = innovator.newItem("ucm_ChangeOrder");

    activeChangeOrderFilter.setProperty("is_released", "0");
    var action = new Aras.UnifiedChangeManagement.GetAffectedItemsInUcmChange(
        "ucm_ChangeOrder_AffectedItem",
        activeChangeOrderFilter,
```

```
        $"'{string.Join("','", idsToCheck)}'",
        IncludeReleasedFilterForRelatedItem: false,
        Select: "id,source_id(keyed_name),related_id(config_id,itemtype,keyed_name)",
        MaxRecords: null
    );
    Item result = actionProcessor.Process(action);

    if (result.isError() && !result.isEmpty())

    {

        throw new Aras.Server.Core.InnovatorServerException(result.getErrorString());

    }
    return result;

}
private void PopulateConflicts(Item conflictingRelationships, AffectedItemValidationResult validatio

{

    for (int i = 0; i < conflictingRelationships.getItemCount(); i++)

    {

        Item conflictingRelationship = conflictingRelationships.getItemByIndex(i);

        string activeChangeOrderName = conflictingRelationship.getPropertyAttribute("source_id", "ke

        Item conflictingAffectedItem = conflictingRelationship.getRelatedItem();

        string configId = conflictingAffectedItem.getProperty("config_id");
        // Cache item metadata
        if (!validationResult.ItemMetadataByConfigId.ContainsKey(configId))

        {

            string itemName = conflictingAffectedItem.getProperty("keyed_name");

            string itemTypeId = conflictingAffectedItem.getProperty("itemtype");

            validationResult.ItemMetadataByConfigId[configId] = new AffectedItemMetadata(itemName, i

        }
        var metadata = validationResult.ItemMetadataByConfigId[configId];
        // Get or create change order entry

        if (!validationResult.ConflictsByChangeOrderAndType.TryGetValue(

            activeChangeOrderName,

            out Dictionary<string, List<string>> conflictsByItemType))

        {

            conflictsByItemType = new Dictionary<string, List<string>>();
```


```
            validationResult.ConflictsByChangeOrderAndType[activeChangeOrderName] = conflictsByItemT

        }
        // Get or create item type entry

        if (!conflictsByItemType.TryGetValue(metadata.ItemTypeId, out List<string> conflictingConfig

        {

            conflictingConfigIds = new List<string>();

            conflictsByItemType[metadata.ItemTypeId] = conflictingConfigIds;

        }
        conflictingConfigIds.Add(configId);

    }

}
private string BuildConflictErrorMessage(AffectedItemValidationResult validationResult)

{

    Dictionary<string, string> itemTypeLabelsByTypeId = GetItemTypeLabels(

        validationResult.ConflictsByChangeOrderAndType.Values

            .SelectMany(conflictsByType => conflictsByType.Keys)

            .Distinct());
    Aras.Server.Core.Abstractions.IErrorLookup errorLookup = GetErrorLookup();

    StringBuilder errorMessage = new StringBuilder();
    foreach (var changeOrderEntry in validationResult.ConflictsByChangeOrderAndType)

    {

        string activeChangeOrderName = changeOrderEntry.Key;

        Dictionary<string, List<string>> conflictsByItemType = changeOrderEntry.Value;
        StringBuilder conflictingItemsDescription = new StringBuilder();
        foreach (var itemTypeEntry in conflictsByItemType)
        {
            string itemTypeId = itemTypeEntry.Key;

            List<string> conflictingConfigIds = itemTypeEntry.Value;
            string itemTypeLabel = itemTypeLabelsByTypeId[itemTypeId];

            IEnumerable<string> itemNames = conflictingConfigIds

                .Select(configId => validationResult.ItemMetadataByConfigId[configId].ItemName);
            string formattedItemGroup = FormatItemGroup(itemTypeLabel, itemNames);

            conflictingItemsDescription.Append($"{formattedItemGroup}, ");

        }
        // Remove trailing comma and space
```

```
        conflictingItemsDescription.Remove(conflictingItemsDescription.Length - 2, 2);
        string errorLine = errorLookup.Lookup(

            "ucm_ItemIsInActiveChange",

            activeChangeOrderName,

            conflictingItemsDescription.ToString());

        errorMessage.AppendLine(errorLine);

    }
    // Remove trailing newline

    errorMessage.Remove(errorMessage.Length - Environment.NewLine.Length, Environment.NewLine.Length

    return errorMessage.ToString();

}
private Dictionary<string, string> GetItemTypeLabels(IEnumerable<string> itemTypeIds)

{

    Dictionary<string, string> labelsByTypeId = new Dictionary<string, string>();
     if (itemTypeIds == null || !itemTypeIds.Any())

    {

        return labelsByTypeId;

    }
    Item itemTypesQuery = innovator.newItem("ItemType", "get");

    itemTypesQuery.setAttribute("select", "id,name,label");

    itemTypesQuery.setPropertyCondition("id", "in");

    itemTypesQuery.setProperty("id", $"'{string.Join("','", itemTypeIds)}'");
    Item itemTypesResult = itemTypesQuery.apply();

    if (itemTypesResult.isError())

    {

        throw new Aras.Server.Core.InnovatorServerException(itemTypesResult.getErrorString());

    }
    for (int i = 0; i < itemTypesResult.getItemCount(); i++)

    {
        Item itemType = itemTypesResult.getItemByIndex(i);

        string typeId = itemType.getProperty("id");

        string label = itemType.getProperty("label", itemType.getProperty("name"));
```

```
        labelsByTypeId[typeId] = label;

    }
   return labelsByTypeId;

}
private static string FormatItemGroup(string itemTypeLabel, IEnumerable<string> itemNames) =>

    $"{itemTypeLabel} ('{string.Join("', '", itemNames)}')";
private static string ExtractConfigIdFromProperty(Item affectedItem, string propertyName)

{

    Item propertyItem = affectedItem.getPropertyItem(propertyName);

    if (propertyItem != null && !propertyItem.isEmpty())
    {
        return propertyItem.getID();

    }
    string propertyValue = affectedItem.getProperty(propertyName);

    if (!string.IsNullOrEmpty(propertyValue))

    {

        return propertyValue;

    }
    return null;

}
internal virtual Aras.Server.Core.Abstractions.IErrorLookup GetErrorLookup()

{

    return (serverConnection as Aras.Server.Core.IOMConnection)?.CCO?.ErrorLookup;]]></method_code>
```

5. Click "Save" to save the method.

Configuring Server Events on Item Types

Configure the method to run on server events for Item Types that use legacy change management (Simple ECO, Express ECO, etc.).

1. **Log in with Administrator privileges.**
   - Sign in as a member of the **Administrators** group.
2. **Navigate to Item Types.**
   - Navigate to Administration → Item Types.
   - Open the Item Type (e.g., "Simple ECO").
   - Click "Edit" to enable editing.
3. **Open the "Server Events" tab.**
4. **Click "Create" to add a new server event.**
5. **Configure Server Events.**

- Add onAfterAdd event.

Field	Description	Value
Name	Identifier for the server event	ucm_NotInAnyActiveChangeOrder
Method	Method to execute	ucm_NotInAnyActiveChangeOrder
Event	Server event trigger	onAfterAdd
Sort Order	Execution order	128

- Add onBeforeUpdate event.

Field	Description	Value
Name	Identifier for the server event	ucm_NotInAnyActiveChangeOrder
Method	Method to execute	ucm_NotInAnyActiveChangeOrder
Event	Server event trigger	onBeforeUpdate
Sort Order	Execution order	256

6. **Click "Save" to save the Item Type configuration.**

Important

Repeat steps 2–6 for each Item Type where validation is needed (Simple ECO, Express ECO, etc.).

Performance Considerations

Important

Turning on this validation can affect performance when you add or update many Affected Items at once. The system uses the **GetAffectedItemsInUcmChange** action to check for active Change Orders, which triggers additional database queries. After enabling the validation, keep an eye on system performance—especially during operations that involve large numbers of Affected Items.