Unified Change Management

Release 30

Last Modified: 04/24/2026

Copyright Information

Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Phone: 978-691-8900

Notice of Rights

Notice of Liability

This document is provided for informational purposes only, and the contents hereof are subject to change without notice. The information contained in this document is distributed on an "as is" basis, without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement. Aras shall have no liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this document or by the software or hardware products described herein.

Table of contents

Release Notes

Platform Support Matrix

The Aras Unified Change Management – Installation Guide lists the software required:

- Aras Innovator Release 30.
- Aras Innovator Release 31.
- Aras Innovator Release 32.
- Aras Innovator Release 33.
- Aras Innovator Release 34.
- Aras Innovator Release 35.
- Aras Innovator Release 36.
- Aras Innovator Release 37.

Key Enhancements and Known Issues

Key Enhancements in Aras Unified Change Management Release 30

The following key features are available in the Aras Unified Change Management Release 30

Change Process Coverage

Aras Unified Change Management supports the full lifecycle of a change, including creation, review, approval, implementation, and closure, along with item release, revision, and obsolescence, with streamlined execution through Change Requests and Change Orders

Affected Items and Impact Analysis

Aras Unified Change Management allows the capture of items to be changed and enables users to perform impact checks by leveraging Where Used and relationship browsing.

Supporting Data and Files

Both Change Requests and Change Orders can capture supporting data and related files. This ensures that all business data, technical references, and documentation needed to evaluate or implement a change are maintained in one place for transparency and traceability.

Change Tasks

Change Tasks add value by breaking significant changes into manageable actions, assigning clear ownership to individuals or teams, and providing status visibility within the Change Order. They reduce execution risk by preventing missed steps and creating full traceability of who did what, and when.

Collaboration and Approvals

Aras Unified Change Management enables team collaboration with defined roles, responsibilities, and signoff tracking.

Dynamic Workflow Assignments

Dynamic Workflow Task Assignment automatically routes tasks to the right users or roles based on real-time business logic and item context. Instead of relying on static workflows, assignments adjust dynamically based on an organization’s defined rules.

Centralized Change Control

Aras Unified Change Management manages multiple item types within a single environment, with configurable change-controlled item types.

Flexible Configuration

Workflows, transitions, and validations can be tailored to align with specific business processes. This flexibility reduces manual effort, accelerates change execution, and ensures that change management aligns closely with organizational requirements.

Known Issues in Aras Unified Change Management Release 30

Issue #	Description	Workaround
IH-6272	Change Tasks are initiated by selecting one or more affected items from the Change Order. On the Change Task creation form, columns showing information about the items are not displayed until the task has been created. Likewise, when adding additional affected items from the Change Order to the Change Task, their information columns are not displayed until the task is saved.	All information is displayed after the Change Task is created and saved, as expected.
IH-4524	“Changes Pending” flag is cleared on an item when versioned if it was set by Change Order’s automation.	Active changes in UCM are displayed on the Changes tab of Affected Items.
IH-977	The Promote button is not yet hidden from Change Order, Change Request and Change Task forms.	Clicking the Promote button does not promote the Change. Use workflow signoff as intended.
IH-3486	When adding a Change Request to a Change Order, the type-in field does not suggest possible values.	Use the search dialog.
IH-7472	In the Change Process Definition, the “From State” and “To State” lifecycle values are not restricted to the its Context Item.	

Administrator Guide

Introduction

Purpose

The Administrator Guide outlines the procedures for configuring and customizing the **Aras Unified Change Management** application.

Scope

This document explains how to configure **Change Requests**, **Change Orders**, and **Change Tasks** to support an organization's business needs. It provides detailed guidance on **Change Process Definition**, **Change Actions**, and **Automations** to enable these configurations.

Target Audience

This guide is intended for **Change Managers** and **Change Administrators** who are responsible for the business and system administration of change processes. It explains how to configure Change Requests, Change Orders, and Change Tasks, define the list of change-controlled item types, and set the rules and actions that govern them.

Overview

The **Change Process Definition** serves as the primary configuration item for managing Change Items within the Aras Unified Change Management application. It enables the configurable behavior of specific context Change item types, such as **Change Requests** and **Change Orders**, as well as **Change Tasks** within the Change Orders. Key components associated with the Change Process Definition include:

- **Change Actions**: These define the selectable actions, such as *Release*, *Revise*, or *Obsolete*, for the affected items of a Change Item.
- **Automations**: These enable the configuration of validations and automated actions applied to Change Items and/or their affected items, streamlining and enhancing process efficiency.

Identities and Permissions

Aras Unified Change Management application installs the following identities:

- **Change Authors** – Used in *Can Add* and *Permissions* configurations for Change Orders. This group includes *Change Administrators* and *Change Managers*.
- **Change Managers** – Used in *Permissions* configurations for Change Orders. They can help manage InBasket tasks by taking over or delegating them when needed. They also inherit the permissions of *Change Authors*.
- **Change Administrators** – Used in *Can Add* and *Permissions* configurations for Change Process Definitions, Change Actions, and Automations Item Types. By default, this group includes *Administrators*. They also inherit *Change Authors'* permissions in the pre-configuration.
- **Change Viewers** – Provides read-only access to Change Orders in the TOC. This group includes *All Employees*, *Change Review Board*, *Change Authors* and *Change Verifiers*.
- **Change Requestors** – Used in *Can Add* and *Permissions* configurations for Change Requests. This group includes *Change Authors*.**Change Review Board** – Assigned to workflow activities within Change Orders.
- **Change Verifiers** – Assigned to workflow activities within Change Orders.
- **Change Executor** – A system identity that automatically promotes the lifecycle state of Change Orders, Change Requests, Change Tasks and executes Automations.

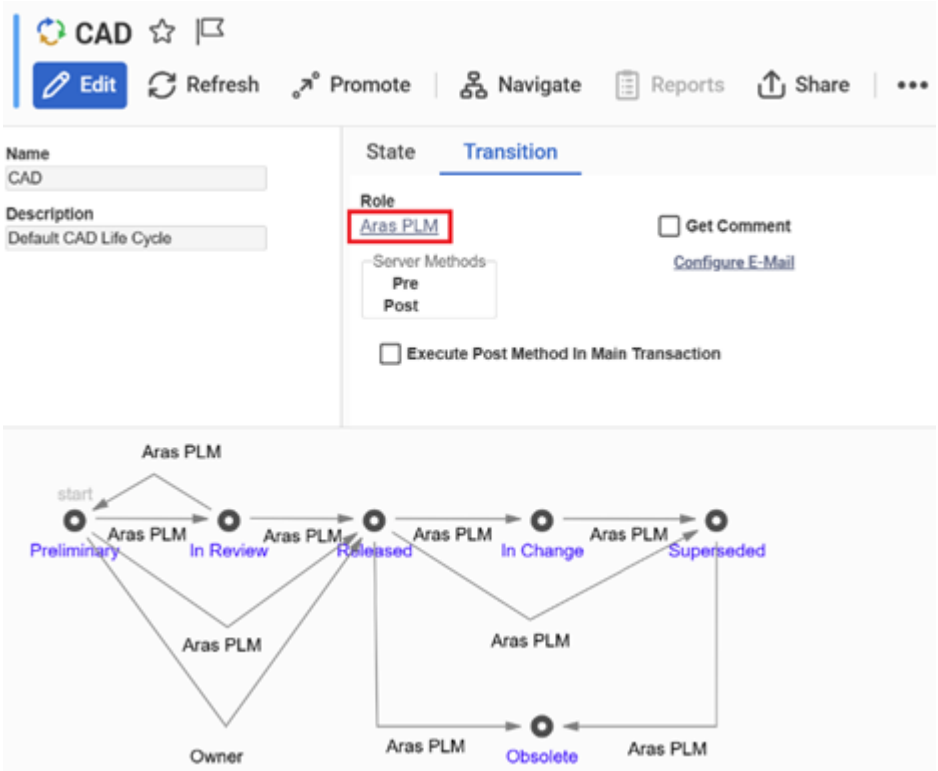
Important

Standard configuration procedures can be used to update permissions for different users, groups, and other identities to meet specific business requirements.

Configuring the Change Executer System Identity

To enable automatic promotion of affected items through Change automations, ensure that the **Change Executor** system identity is included in the role identities assigned to the corresponding lifecycle transitions.

- Review the Item Type's lifecycle map to identify the role(s) assigned to each lifecycle transition and add **Change Executor** to the role(s) responsible for these transitions.



Permissions for Change Process Definition

The following table lists the identities and their permissions for the Change Process Definition:

ItemType	Change Administrators	Change Executor	World
Change Actions	Full access, except “Can Change” permissions	Get, Can Discover	Get, Can Discover
Automations	Full access (except Can Change Access)	Get, Can Discover	-
Change Process Definitions	Full access (except Can Change Access)	Get, Can Discover	-

Permissions for Change Request

The following table lists the identities and their permissions for Change Requests:

Lifecycle State	Workflow Activity	Role	Create	Update	Get	Can Discover	Delete	Show Permissions Warning	Can Change Access
Draft	Start, Prepare Change Proposal & Analyze Change Impact	Creator		+	+	+	+		
		Change Administrators	+ (through Change Requestors)	+	+	+		+	+
		Change Managers	+ (through Change Requestors)	+	+	+		+	+
		Change Viewers			+	+			
		Change Requestors	+		+	+		+	
		Change Executor		+	+	+			
		Team Member		+	+	+			
In Review	Review Change Request	Creator			+	+			
		Change Administrators			+	+		+	+
		Change Managers			+	+		+	+
		Change Viewers			+	+			
		Change Requestors			+	+		+	
		Change Executor		+	+	+			
		Team Member			+	+			
Rejected	Reject	Creator			+	+			
		Change Administrators			+	+		+	+
		Change Managers			+	+			
		Change Viewers			+	+			
		Change Requestors			+	+		+	

		Change Executor			+	+			
		Team Member			+	+			
Approved	Approve	Creator			+	+			
		Change Administrators			+	+		+	+
		Change Managers			+	+			
		Change Viewers			+	+			
		Change Requestors			+	+		+	
		Change Executor			+	+			
		Team Member			+	+			
Canceled	Cancel	Creator			+	+			
		Change Administrators			+	+		+	+
		Change Managers			+	+			
		Change Viewers			+	+			
		Change Requestors			+	+		+	
		Change Executor			+	+			
		Team Member			+	+			
Closed		Creator			+	+			
		Change Administrators			+	+		+	+
		Change Managers			+	+			
		Change Viewers			+	+			
		Change Requestors			+	+		+	
		Change Executor			+	+			
		Team Member			+	+			

Permissions for Change Orders

The following table lists the identities and their permissions for the Change Orders:

Lifecycle State	Workflow Activity	Role	Create	Update	Get	Can Discover	Delete	Show Permissions Warning	Can Char Acce
Draft	Start & Prepare Implementation Plan	Creator		+	+	+	+		
		Change Administrators	+ (through Change Authors)	+	+	+	+	+	+
		Change Authors	+	+	+	+		+	+
		Change Managers	+ (through Change Authors)	+	+	+	+	+	+
		Change Executor		+	+	+			
		Change Viewers			+	+			
		Team Member		+	+	+			
In Review / Audit	Review Implementation Plan / Verify Implementation	Creator			+	+			
		Change Administrators			+	+		+	+
		Change Authors			+	+		+	+
		Change Managers			+	+		+	+
		Change Executor		+	+	+			
		Change Viewers			+	+			
		Team Member			+	+			
In Work	Execute Change Plan	Creator		+	+	+			
		Change Administrators		+	+	+		+	+
		Change Authors			+	+		+	+
		Change Managers		+	+	+		+	+

		Change Executor		+	+	+			
		Change Viewers			+	+			
		Team Member			+	+			
Completed	Complete Change	Creator			+	+			
		Change Administrators			+	+		+	+
		Change Authors			+	+		+	+
		Change Managers			+	+			
		Change Executor			+	+			
		Change Viewers			+	+			
		Team Member			+	+			
Canceled	Cancel	Creator			+	+			
		Change Administrators			+	+		+	+
		Change Authors			+	+		+	+
		Change Executor			+	+			
		Change Managers			+	+			
		Change Viewers			+	+			
		Team Member			+	+			

Permissions for Change Tasks

The following table lists the identities and their permissions for the Change Tasks:

Lifecycle State	Workflow Activity	Role	Create	Update	Get	Can Discover	Delete	Show Permissions Warning	Can Change Access
Not Started	Start, Prepare Change Task & Confirm Change Order is in work	Creator		+	+	+	+		
		Change Authors	+	+	+	+		+	+
		Change Viewers			+	+			
		Change Executor		+	+	+			
		Team Member		+	+	+			
		Team Manager		+	+	+			
In Review	Review Change Task	Creator			+	+			
		Change Authors		+	+	+		+	+
		Change Viewers			+	+			
		Change Executor		+	+	+			
		Team Member		+	+	+			
		Team Manager		+	+	+			
In Progress	Execute Change Task	Creator			+	+			
		Change Authors			+	+		+	+
		Change Viewers			+	+			
		Change Executor		+	+	+			
		Team Member			+	+			
		Team Manager			+	+			
Completed	Complete Change	Creator			+	+			

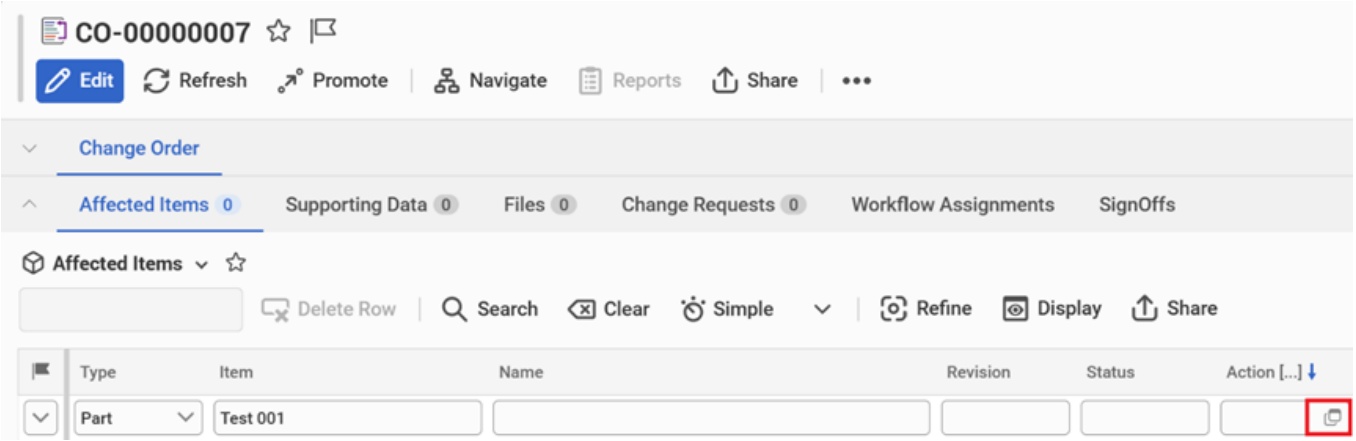
	Task	Change Authors			+	+		+	+
		Change Viewers			+	+			
		Change Executor			+	+			
		Team Member			+	+			
		Team Manager			+	+			
Canceled	Cancel	Creator			+	+			
		Change Authors			+	+		+	+
		Change Viewers			+	+			
		Change Executor			+	+			
		Team Member			+	+			
		Team Manager			+	+			

Configuring Change Action Items

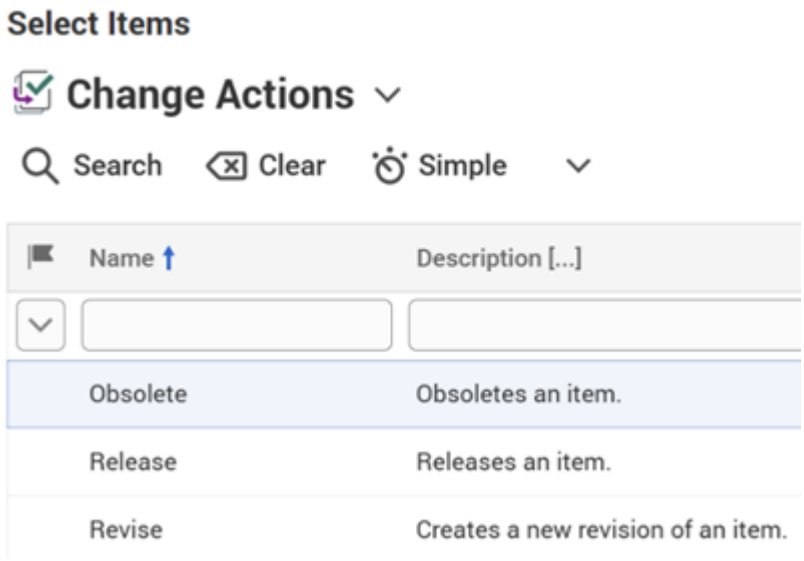
Change Actions specify the operations—like **Release**, **Revise**, or **Obsolete**—that can be performed on affected items of a Change Item. Custom Change Actions can also be created for special operations or business logic. These actions can apply to multiple change-controlled item types, such as **Parts**, **CAD Documents**, and **Documents**.

To view Change Actions for affected items in a Change Order:

- Open the **Change Order** and go to the **Affected Items** tab.
- In the **Action** field of an affected item, click the **pop-up icon** to see available actions.



- Choose an appropriate Change Action on an Item:

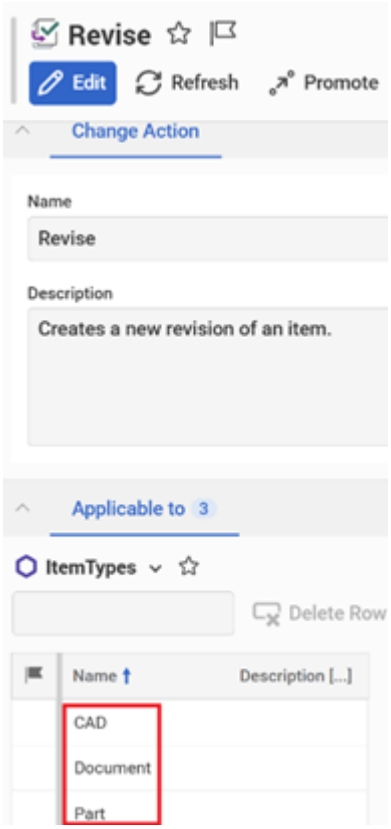


- Validations are performed for the selected action. If a validation fails, an error message is displayed.

Creating a New Change Action

The steps to create a **Change Action** are as follows:

1. As a member of **Change Administrators**, go to **Contents**, expand **Unified Change Management** → **Configuration**, and select **Change Actions**. Click **Create New Change Action**.
2. Enter a **Name** and **Description** for the action.
3. On the **Applicable To** tab, select the item type(s) for which this Change Action is valid.



4. Click **Done**.

Once created, a **Change Action** can be used as a condition in automations. For example, a validation automation can check whether an affected item with a **Revise** action is currently in the **Released** state. The setup and use of Change Actions in automations are managed through automation rules within the **Change Process Definition**. For more information, see **Section 6.2 Manage Automation Rules for a Change Process Definition**.

Configuring Automation items

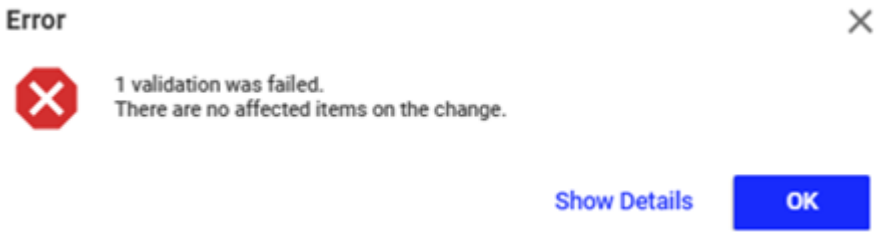
Automations are configurable components that define checks and actions. They can apply to the Affected Items (Relationship) or the **Change Order** itself (*Context Item*).

Each Automation falls into one of **two types**:

- **Validation** – System checks that enforce data quality and business rules:
  - Ensuring a Change Order does not include affected items with missing required properties.
  - Verifying that when an action like **Revise** is selected, the affected item is in the **Released** state.
- **Action** – Automated updates applied to affected items or the Change Order:
  - Automatically changing the lifecycle state of affected items (e.g., from **In Review** to **Released**) after a Change Order's workflow is completed, based on the defined action.
  - Automatically setting the **Actual Completion Date** on the Change Order when its workflow finishes.

Automations improve consistency and efficiency in the change process by reducing manual work and enforcing business rules automatically.

The following is an example of an error message generated during a validation Automation:



Creating an Automation

The steps to create an **Automation** are as follows:

1. As a member of **Change Administrators**, go to the **Contents**, expand **Unified Change Management** → **Configuration**, and select **Automations**. Then click **Create New Automation**.
2. Fill in the key properties:
    - **Method** – The method that contains the business logic for the Automation.
    - **Classification**
        - **Validation** – Ensures compliance with data quality or business rules.
3. **Action** – Executes automated modifications.
4. Click **Done**.

Automation Execution and is_executed Flag

Background

The UCM automation uses an internal is_executed flag on item relationships to track when a change action starts. This flag helps control when you can edit effected items during a Change order and keeps all changes within the proper workflow boundaries.

Some automations run extra steps that are not part of the actual change action. When this happens, the automation needs to set the **skip_is_executed** attribute so it does not accidentally mark the relationship as executed.**When to Use skip_is_executed:**

Use **skip_is_executed**="1" when the automation performs supporting operations that do not represent the execution of the change action itself, such as:

- Setting tracking properties (e.g., has_change_pending).
- Sending notifications.
- Creating audit records.
- Updating metadata unrelated to the change action.

Important

Do NOT use **skip_is_executed** for automations that execute the actual change action, including any operation that represents the irreversible execution of the intended change (e.g. creating new revision).

Implementation Example

The "Set has_change_pending property" automation demonstrates proper use of skip_is_executed.

Configuring Change Process Definition items

The **Change Process Definition** is the primary configuration tool that allows Change Administrators to customize the behavior of Change Items.

For example, a Change Process Definition can specify:

- Automatic validations for Change Orders and their affected items.
- Automatic lifecycle transitions triggered by workflow sign-offs, based on the Item Actions.

Each Change Process Definition is linked to a single context Change Item Type (such as a Change Order).

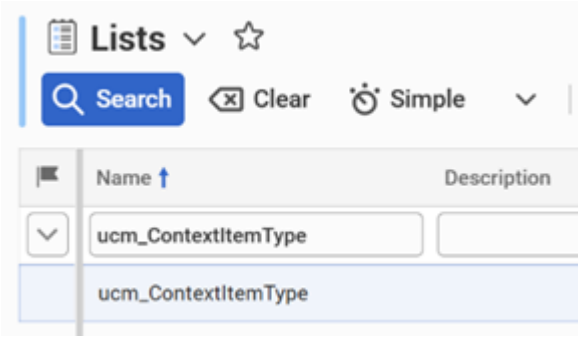
On the **Automation Rules** tab, administrators can associate multiple Automations with a Change Process Definition. These Automations define the validations and actions for the Change Item Type and its affected items, with Change Actions serving as the trigger for each Automation.

and its affected items, with **Change Actions** serving as the trigger for each Automation.

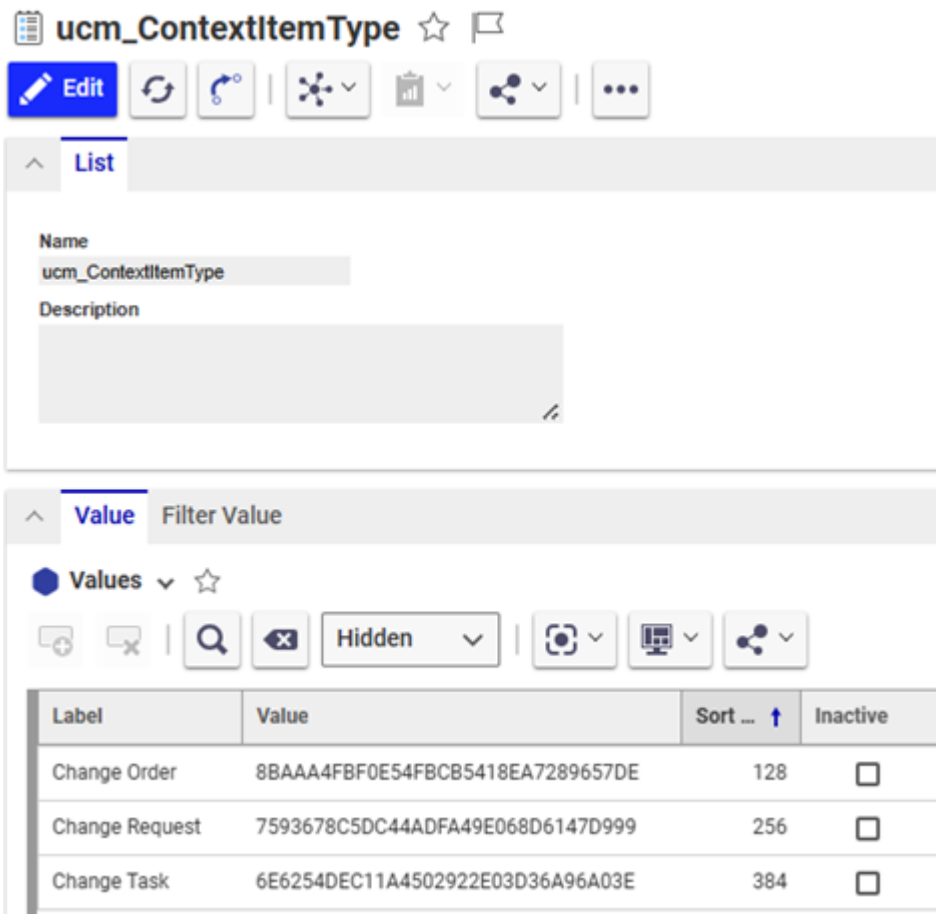
Creating Change Process Definition Item

The steps to create a **Change Process Definition** are as follows:

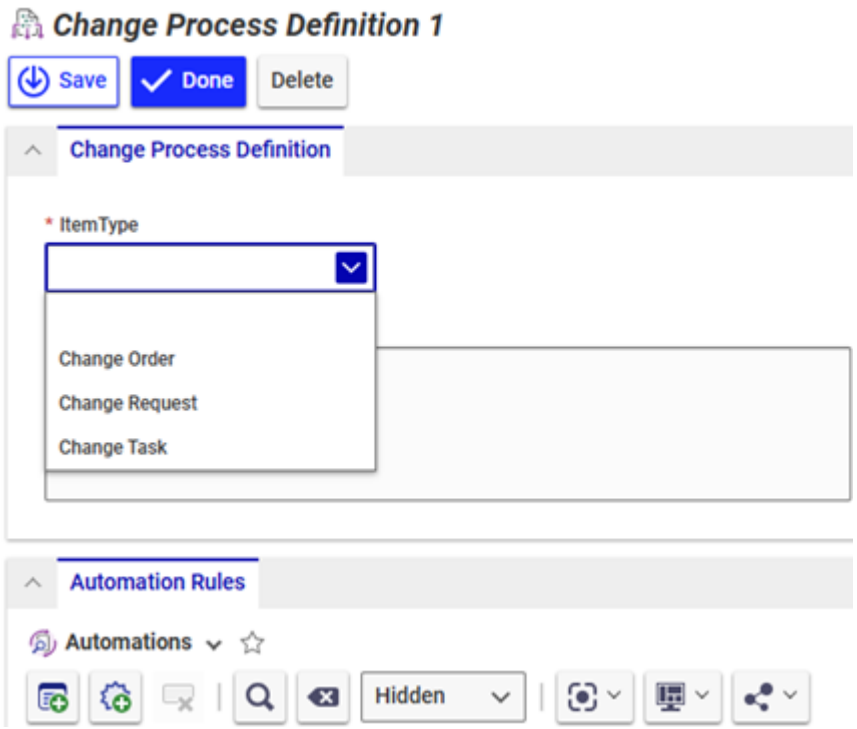
1. As a member of **Change Administrators**, navigate to **Contents** → **Administration** → **Lists** and search for **ucm_ContextItemType**.



2. Open the **ucm_ContextItemType** list item.
3. Update the list of values to add an entry for the Change item.
    - **Label** – Enter a user-friendly name (for example, *Change Order*).
    - **Value** – Enter the corresponding Item Type ID.



4. Navigate to the **Contents**, expand **Unified Change Management** → **Configuration**, and select **Change Process Definitions**. Then click New Change Process Definition.

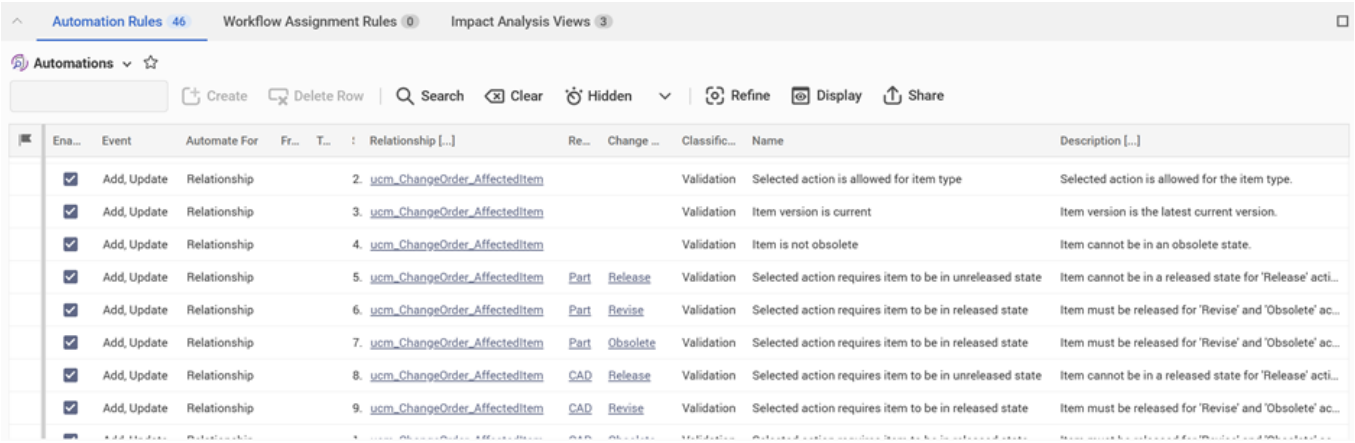


5. Set Item Type to the context Change Item Type where the automation will apply.
6. Enter a **Description**, then click **Save**.

Managing Automation Rules for Change Process Definition

The steps to create, add, or modify Automations for a Change Process Definition are as follows:

1. Go to the **Contents** → **Unified Change Management** → **Configuration** → **Change Process Definitions**. Search and open it for edit.
2. Go to the **Automation Rules** tab and choose one of the following:
   - **Add Automations** – Attach one or more existing Automations to the relationship.
   - **Create Automation** – Build and add a new Automation to the relationship.



The automations tabs are: **Classification**, **Name** and **Description**. Remaining tabs are the conditions triggering the automation.

3. **Fill** or **modify** the following columns as appropriate:

Field	Description	Details / Options
Enabled	Activates or deactivates the automation.	✔ Checked = active ✘ Unchecked = inactive
Event	Defines what triggers the automation on the context item (e.g., Change Order) or relationship (e.g., Affected Items).	• **Add** – Runs when the context item or a relationship is added. When an item is copied, all automations configured with the ****Add**** event will execute on the copied item. • **Promote** – Runs during a lifecycle transition of the context item. Example: set the Actual Completion Date when a Change Order is completed. ⚠ Promote works only with Context Items, not Relationships. • **Update** – Runs when the context item or a relationship is modified. • **Remove** – Runs when a relationship is removed.

Automate For	Defines which item triggers the automation.	
From State / To State	Used only for **Promote** events. Defines the current and target lifecycle state of the Context Item.	• Specific lifecycle states can be selected. To apply to all states, select "**Any**". Example: **In Review → To Released. Any → To Released.** • The list of values is retrieved from the **Filter Value** tab within the **ucm_TargetLifeCycleState** list.
Relationship	**Required** if **Automate For** = **Relationship**.	• Specify the **Relationship Type** of the Context Change Item on which the automation applies.
	Optional when used for **Promote** events.	• Specify the **Relationship Type** linked to the Context Change Item Type on which the automation applies.
Related Item Type	Optional, only used when **Automate For** = Relationship.	• Specify the **Item Type of related items** for the selected relationship on which the automation applies. • If left empty, the automation applies to all related item types for the selected relationship.
	Optional when used for **Promote** events.	• Specify the **Item Type of related items** for the Context Change Item's selected relationship on which the automation applies. • If left empty, the automation applies to all related item types for the selected relationship.
Related Event	The Change Action(s) that trigger the automation.	• Preconfigured actions are **Release**, **Revise**, **Obsolete**.
	Optional when used for **Promote** events.	• Specify the **Action** for the selected relationship on which the automation applies when the Context Change Item is promoted.

The columns **Classification, Name, Method, Description,** and **Automation Type** describe the properties of the selected Automation.

- **Method** – Contains the implementation code that executes the automation (refer to **Section 5** Configuring Automation items for details.)
- **Sequence** – Defines the order in which Automation rules are executed.

Important

The automation method may need to be modified to accommodate custom Change Actions, processing logic, lifecycle states, and other extensions. Refer to **Section 8** *Configurations for Extending Change-Controlled Items*.

Dynamic Workflow Assignments

Overview

Dynamic Workflow Assignments automate task routing within Change Management workflows. Rather than manually defining assignees for each workflow activity, administrators configure rules that determine the correct assignees based on business logic and workflow context. When a workflow is created, assignments are automatically resolved and applied.

Key Benefits:

- Decreases administrative effort and reduces manual errors by automating assignments
- Scales easily as teams or processes evolve
- Flexible logic tailored to organizational needs

Dynamic Workflow Assignments include three core components:

1. **Workflow Assignment Rules:** Reusable definitions of assignment logic (via server methods).
2. **Change Process Definition Configuration:** Links rules to specific workflows or activities within a Change Process Definition.
3. **Rule Execution Engine:** Processes and executes rules when workflows are initiated

Assignments are created at workflow creation time, ensuring activities are ready to execute before any user interaction.

Key Considerations:

- No pre-configured rules are provided. Aras Unified Change Management gives you full flexibility to design Workflow Assignment Rules that match your organization's specific processes. Administrators and/or developers create all rules.
- Rules execute once during workflow creation. Changes to teams, items, or structures after that point do not trigger reassignment.

Example Use Cases

Important

These examples require custom assignment logic – not included out of the box.

Scenario	Example Logic
Based on affected items	Assign part owners when their parts appear on a change

Based on business unit/region	Route wind turbine changes to the Denmark team
Based on change characteristics	Escalate urgent changes to senior leadership

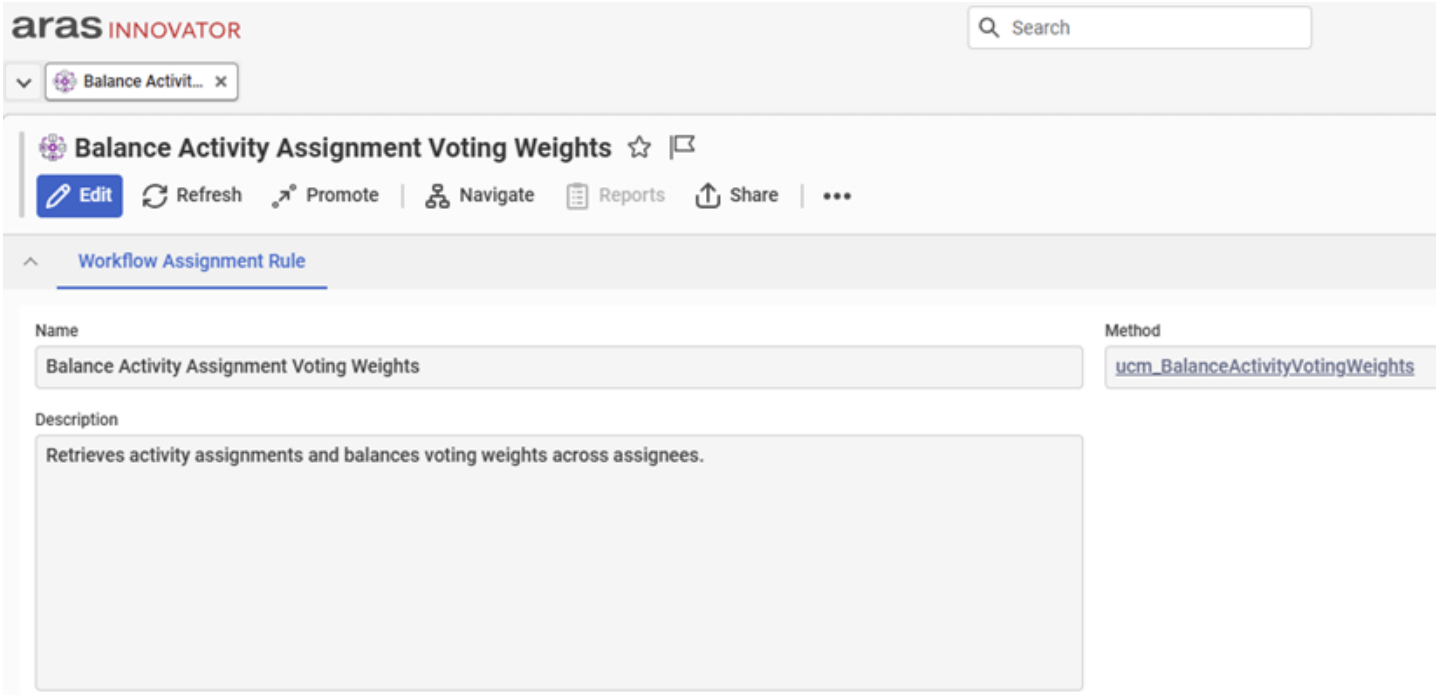
Workflow Assignment Rules

Workflow Assignment Rules define reusable logic that determines who receives workflow tasks.

Creating Workflow Assignment Rules

You must be a member of the Change Administrators Idenittiy to create a Workflow Assignment Rule. To create a Workflow Assignment Rule:

1. Navigate to Table of Contents → Unified Change Management → Configuration → Workflow Assignment Rules.
2. Select Create New
3. Fill in the required fields:
    - **Name:** The name of the rule (e.g. – “Assign Part Owner to Review Activity”)
    - **Method:** The server method containing assignment logic.
      Description: – (Optional) A brief description of the rule’s purpose and when/where the rule should be used.
4. Click **Done** to save the rule.



Rule Method Requirements

Workflow Assignment Rule methods must follow specific requirements to receive context information and create assignments correctly.

Input

Rules run against:

- A Workflow Process (Target = “Workflow”) or
- An Activity (Target = “Activity”)

Context is provided via:

```
// Context attributes passed by the rule execution engine
 string contextTypeId = this.getAttribute("context_type_id"); // ItemType ID of CPD-controlled item
 string contextItemId = this.getAttribute("context_item_id"); // Item ID of CPD-controlled item
```

This allows rules to inspect the source Change Item (e.g., CO, CR, CT) and its data.

Expected Behavior

Rule methods must:

1. Load context item (if needed)
2. Check existing assignments
3. Determine correct assignees based on business logic
4. Add, update, or remove Activity Assignments
5. Return success or error

Return Values

Rule methods must return either a success result or an error:

Success:

```
return this; // Return the original item
// Or
return inn.newResult("Created 3 assignments successfully");
```

Error:

```
return inn.newError("No manager found for user"); // SOAP Fault - recommended
// Or
throw new Exception("Database connection failed"); // Exception - critical errors only
```

See Section 7.4.2 for detailed error handling strategy and best practices.

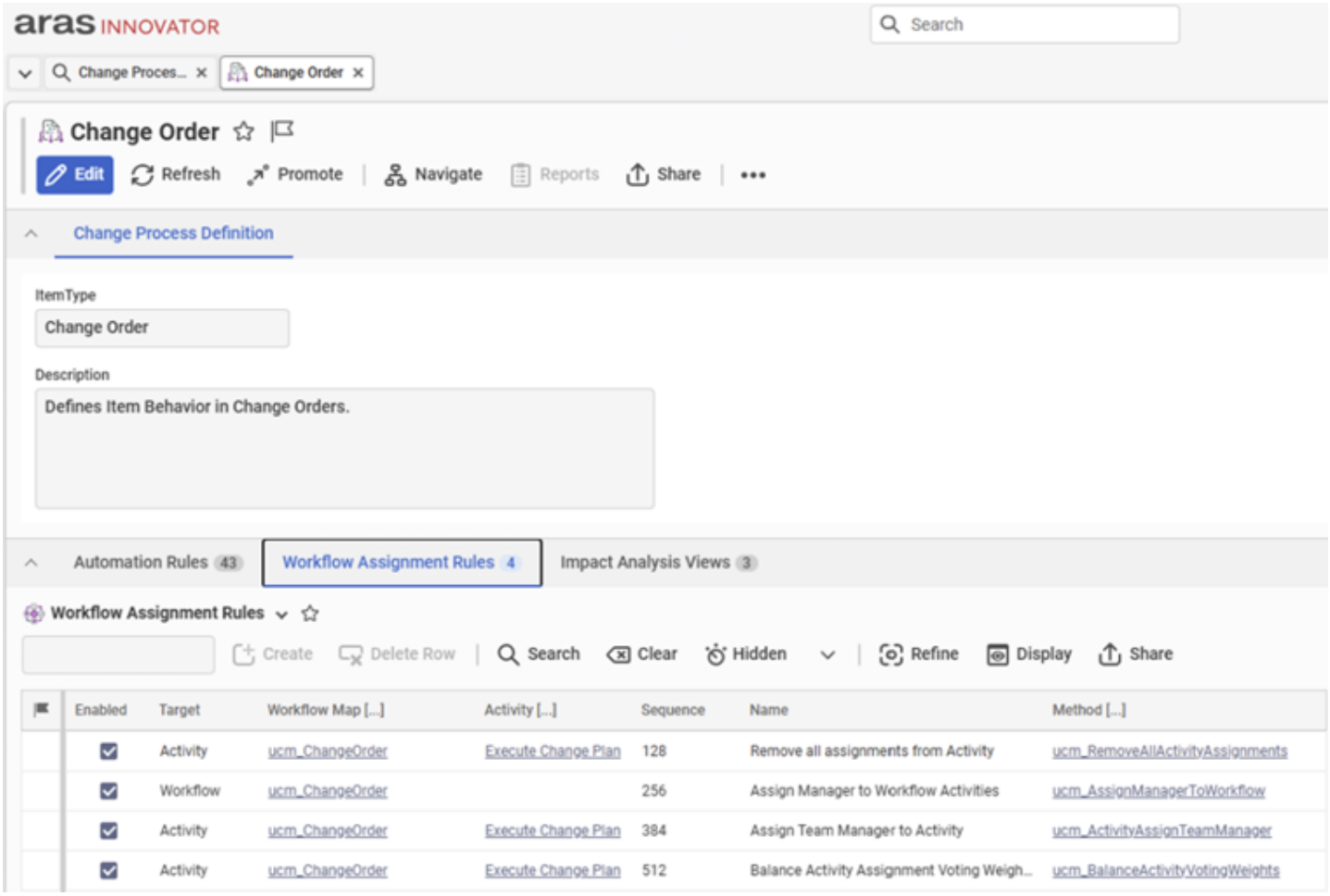
Configuring Rules on Change Process Definitions

Workflow Assignment Rules must be configured within Change Process Definitions to define when and where they execute.

Relationship Grid Configuration

You must be a member of the Change Administrators Idenittiy to configure Workflow Assignment Rules on a Change Process Definition. To configure Workflow Assignment Rules on a Change Process Definition:

1. Navigate to Table of Contents → Unified Change Management → Configuration → Change Process Definitions.
2. Search for and open the Change Process Definition you want to configure.
3. Select the Workflow Assignment Rules tab.
4. Add or create rules.
5. Configure the rule execution details for each rule:



Relationship Grid Columns

The following table describes the columns in the Workflow Assignment Rules relationship grid:

Field	Description	Details / Options
Enabled	Enables or disables the rule	☑ Checked = rule executes ☐ Unchecked = rule is skipped Default: Checked
Target	Defines whether the rule applies to the entire workflow or a specific activity	**Workflow** – Rule applies to the selected workflow **Activity** – Rule applies only to the specified activity
Workflow Map	Specifies which workflow map the rule applies to	Select any Workflow Map that belongs to the CPD target Item Type. **Required** for both Workflow and Activity targets.
Activity	Specifies which activity the rule applies to (only applicable when	Select an Activity Template from the chosen Workflow Map.

	Target = “Activity”)	Only applicable when Target = “Activity”. Should be empty when Target = “Workflow”.
Rule	The Workflow Assignment Rule to execute	Select from available Workflow Assignment Rules. The rule’s method will be invoked during workflow creation.
Sequence	Execution order (ascending)	Integer value determining execution order. Lower numbers execute first (e.g., 128, 256, 384). Rules with the same sequence may be executed in any order.

Target Behavior

Target	Behavior	When to Use
Workflow	Executes once per workflow	Assign global roles or enforce workflow-wide logic
Activity	Executes for each matching Activity item	Assign specialists to specific review/approval steps

Important

If the selected Workflow Map contains multiple activities with the same name as the selected Activity Template, the rule will be executed for **all matching activities**. Activity matching is based on the Activity Template name, not a specific instance.

Multiple Activity Matching:

If your Workflow Map contains multiple activities with the same Activity Template name, the rule will be executed for **all activities matching that name**. This behavior is useful for:

- Applying the same assignment logic to all instances of a particular activity type
- Implementing consistent assignments across parallel workflow paths
- Reducing configuration overhead when multiple activities require identical assignment rules

Example Scenario: A workflow contains three “Technical Review” activities in parallel paths for different product components. A single activity-level rule targeting “Technical Review” will execute for all three activities, assigning the appropriate technical experts to each.

Validation Constraints

The relationship grid enforces the following constraints:

1. **Rule Target Uniqueness**: No duplicate combinations of Rule + Target + Workflow Map + Activity are allowed. Each unique configuration can appear only once in the grid.

2. **Workflow Map Compatibility**: The selected Workflow Map must belong to the CPD target Item Type. The system validates this during configuration.
3. **Activity Existence**: When Target = “Activity”, the specified activity must exist in the selected Workflow Map.
4. **Sequence Order**: Sequence determines execution order. Plan your sequence values carefully to ensure rules are executed in the correct order (e.g., use increments of 10: 10, 20, 30 to allow for future insertions).

Rule Execution

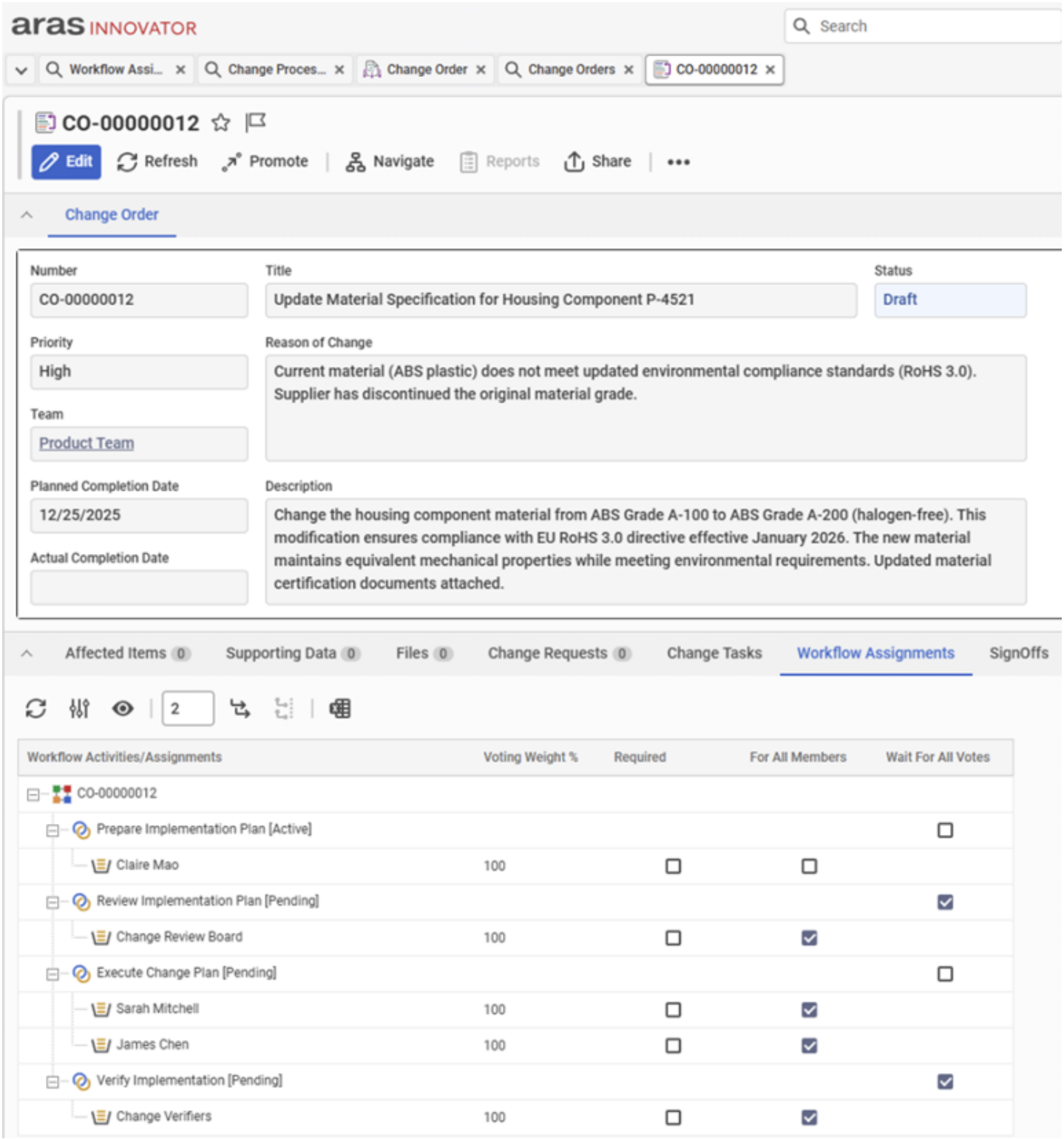
Workflow Assignment Rules execute automatically when a workflow process is created, ensuring assignments are established before users interact with the workflow.

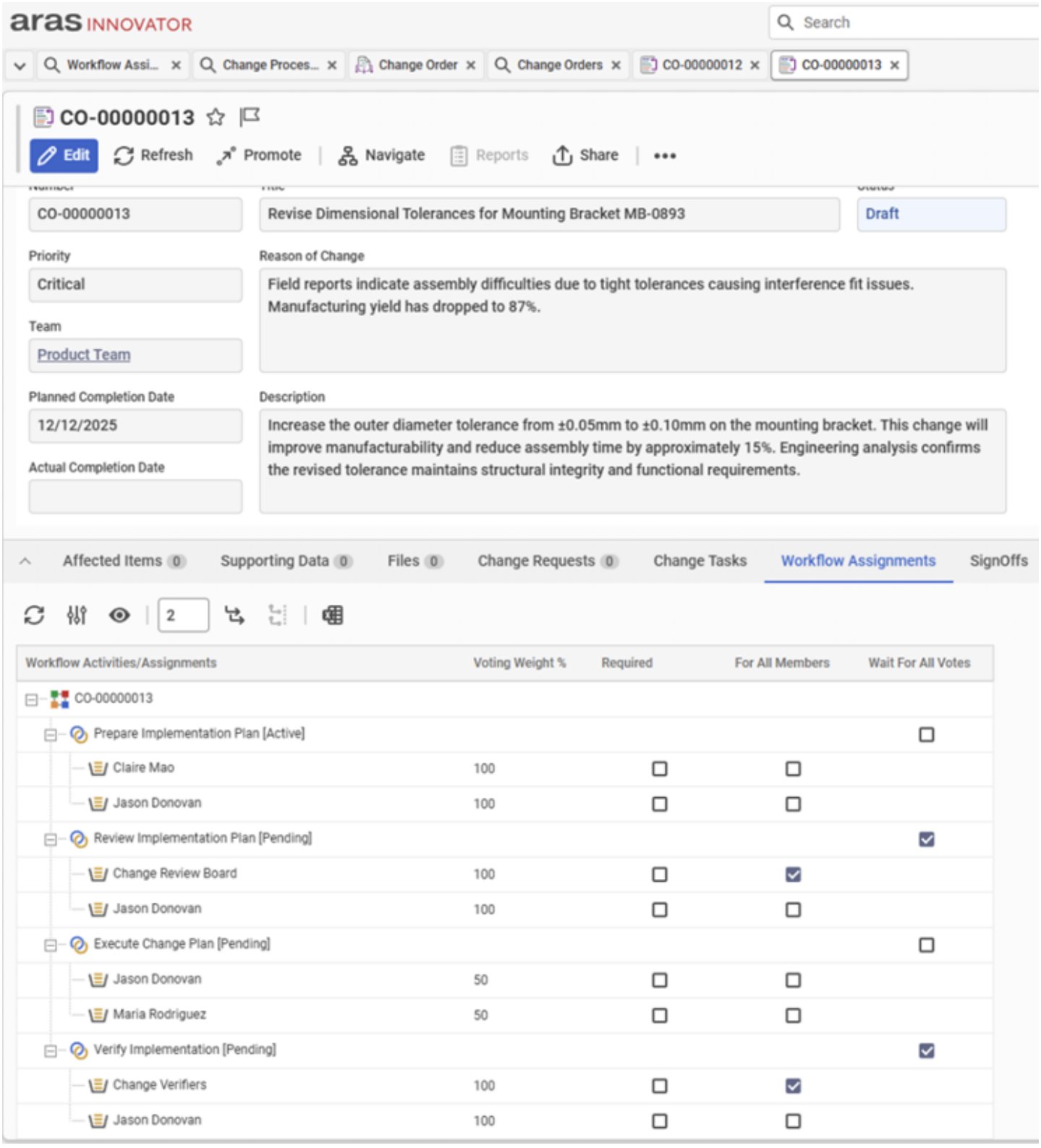
Important

All assignments must be valid for Workflow Process to be able to proceed correctly.

Execution Flow

1. **Workflow Process Creation**: A CPD-controlled item (Change Order, Change Request, Change Task, etc.) creates a Workflow Process based on the default Workflow Map.
2. **Event Trigger**: The Workflow Process onAfterAdd event fires after the workflow is created.
3. **Rule Retrieval**: The dynamic assignment engine queries the related Change Process Definition and retrieves all enabled Workflow Assignment Rules configured for the workflow.
4. **Sequence Ordering**: Rules are sorted by their Sequence value in ascending order (lowest to highest).
5. **Rule Filtering**: The system identifies which rules apply based on Target configuration:
    - **Workflow** target rules: Execute once with the Workflow Process as context
    - **Activity** target rules: Execute for each matching Activity in the workflow
6. **Context Preparation**: For each rule execution, the system adds context attributes to the target item:
    - @context_type_id: Item Type ID of the CPD-controlled item (e.g., Change Order)
    - @context_item_id: Item ID of the CPD-controlled item
7. **Rule Method Invocation**: Each rule's method is called with the prepared context item:
    - Method receives Workflow Process or Activity as this
    - Method can access context attributes to query the source item
    - Method creates or modifies Activity Assignment items as needed
8. **Assignment Creation**: Rule methods modify Activity Assignment items according to custom business logic.
9. **Error Collection**: The system tracks execution results for each rule (see Section 7.4.2 for error handling details).
10. **Completion**: Workflow becomes active with assignments established or fails if errors occurred.





Context Information

Rule methods receive context information through XML attributes on the target item:

```
// These attributes are set by the rule execution engine
string contextTypeId = this.getAttribute("context_type_id");
string contextItemId = this.getAttribute("context_item_id");

// Example: Loading the source Change Order
Innovator inn = this.getInnovator();
Item sourceItem = inn.newItem();
sourceItem.setAction("get");
sourceItem.setAttribute("typeId", contextTypeId);
sourceItem.setID(contextItemId);
sourceItem = sourceItem.apply();

// Now you can access Change Order properties, affected items, etc.
```

This context enables rules to make assignment decisions based on

- Affected items and their properties (owners, classification, etc.)
- Supporting data attached to the change
- Change Order properties (priority, description, managed_by_id)
- Related workflow or activity information

Error Handling Strategy

The rule execution engine supports two types of errors with different behaviors:

Error Type	Behavior	When to Use
SOAP Fault	Continue running other rules; all warnings shown at end	Business condition failures (e.g., user not found)
Exception	Stop execution immediately; workflow creation fails	System failures (e.g., null references)

Error Logging

All rule execution is logged with “WAR:” prefix:

- WAR: Executing rule [RuleName] for [Target]
- WAR: Rule [RuleName] completed in [X]ms
- WAR: Warning - Rule [RuleName] returned error: [Message] (SOAP Fault)
- WAR: Error - Rule [RuleName] threw exception: [Message] (Exception)

Configuring Log Level for Workflow Assignment Rules:

To enable detailed logging for Workflow Assignment Rules execution, add the following configuration to the server’s appsettings.json file:

```
"LoggerConfiguration": {
"MinimumLevel": {
"Default": "Fatal",
"Override": {
"Aras.UnifiedChangeManagement.WorkflowAssignmentRules": "Debug"
}
}
}
```

Log levels available:

- **Debug**: Logs all WAR execution details, timing, and results
- **Information**: Logs rule start and completion
- **Warning**: Logs only SOAP Fault errors
- **Error**: Logs only Exceptions
- **Fatal**: Logs only critical Aras Innovator failures

Important

The “Fatal” log level is available for detailed logging of Workflow Assignment Rules only in Aras Innovator Release 34 and higher.

Team Expansion

When workflow activities are assigned to Team roles, the system automatically expands team membership to create individual assignments for member of the team.

Team Assignment Resolution

Team Membership Expansion:

- Activities may be assigned to a single user, a group, or a Team role.
- If a Team role is used, the system looks up all members of that Team.
- Each member receives their own Activity Assignment.

Resolution Timing:

- Team expansion occurs when the workflow is created, before any Workflow Assignment Rules (WARs) execute
- By the time WARs run, any Team roles assigned to activities have already been expanded into individual Identity assignments

Important

If your WAR business logic needs to assign Team roles, the rule method must explicitly query the Team and create assignments for individual member Identities. The system will **not** automatically expand Team roles assigned by WAR methods.

What Administrators Should Expect

Static Membership

- Team membership used for assignments is locked in when the workflow starts.
- Changes to the Team afterward do not update existing assignments.
    - New members won't receive assignments already created.
    - Removed members keep the assignments they had.

Assignment Items Always Reference Identities

- Activity assignments always reference Identity items (individual users or groups), never directly to Team roles.
- To filter by specific team role (e.g., "Team Manager", "Team Member"), query the team_role property in the Team Identity relationship

Best Practices

Use Teams for Consistent Patterns:

- Define teams for recurring assignment patterns (e.g., “Engineering Managers”, “Quality Reviewers”)
- Reference teams in rule logic rather than hard-coding individual users
- Maintain team membership centrally for easier administration

Consider Team Size Impact:

- Large teams create many individual assignments, which may impact workflow complexity
- Review team size before using teams in high-frequency workflows
- Consider using dynamic assignment rules to filter team members based on additional criteria

Document Team Requirements:

- Include team membership requirements in rule descriptions
- Document which teams are expected to exist for rules to function correctly
- Communicate team structure dependencies to Change Administrators

Query Teams in Rules Sample:

- If rules depend on Team roles, query the team during the WAR execution and create assignments explicitly

```
// Example: Query Team members with specific role and create assignments
Innovator inn = this.getInnovator();
string activityId = this.getID(); // Assuming Target = Activity
// Apply business logic to determine Team ID, e.g. team assigned to context item
// ... your custom logic here ...
Item team = inn.newItem("Team", "get");
team.setID(teamId);
team.setAttribute("select", "id ");
// Query Team Identity relationships filtered by Team Manager role
Item teamIdentityRel = team.createRelationship("Team Identity", "get");
teamIdentityRel.setAttribute("select", "related_id");
Item teamRole = teamIdentityRel.createPropertyItem("team_role",
"Identity", "get");
teamRole.setProperty("name", "Team Manager");
```

```
team = team.apply();

if (team.isError())
{
return inn.newError("Failed to get Team: " + team.getErrorDetail());
}

// Iterate through Team Identity relationships to get Team Manager Identities
Item teamIdentities = team.getRelationships("Team Identity");
int managerCount = teamIdentities.getItemCount();

if (managerCount == 0)
{
return inn.newResult("No Team Managers found in team");
}

for (int i = 0; i < managerCount; i++)
{
Item teamIdentityRel = teamIdentities.getItemByIndex(i);
string managerIdentityId = teamIdentityRel.getProperty("related_id");

// Create assignment for this Team Manager Identity
Item assignment = inn.newItem("Activity Assignment", "add");
assignment.setProperty("source_id", activityId);
assignment.setProperty("related_id", managerIdentityId);
assignment.setProperty("voting_weight", "100");
assignment = assignment.apply();

if (assignment.isError())
{
return inn.newError("Failed to assign Team Manager: " +
assignment.getErrorDetail());
}
}
```

```
 return inn.newResult("Assigned " + managerCount + " Team Manager(s) to
activity");
```

Creating Custom Assignment Rules

Custom Workflow Assignment Rules require developing server methods that implement assignment logic specific to your business requirements.

Development Requirements

Method Requirements

Workflow Assignment Rule methods must satisfy the following requirements:

Method Type: VB/C# server method

Method Context:

- Receives Workflow Process item (Target = “Workflow”)
- Or receives Activity item (Target = “Activity”)

Required Attributes:

- Method can access context_type_id and context_item_id attributes
- These provide the source Change Item information

Expected Operations:

- Query context item to gather assignment decision data
- Query existing Activity Assignment items if needed
- Modify Activity Assignment items with appropriate properties
- Return success result or error

Permissions:

- Method must have appropriate execution permissions

Code Structure

Basic structure for a Workflow Assignment Rule method:

// 1. Get Innovator context

```
Innovator inn = this.getInnovator();
```

```
// 2. Resolve context item
 string contextTypeId = this.getAttribute("context_type_id");
 string contextItemId = this.getAttribute("context_item_id");

// Validate context
 if (string.IsNullOrEmpty(contextTypeId) ||
string.IsNullOrEmpty(contextItemId))
 {
 return inn.newError("Unable to determine context item");
 }

// 3. Load context item
 Item contextItem = inn.newItem();
 contextItem.setAction("get");
 contextItem.setAttribute("typeId", contextTypeId);
 contextItem.setID(contextItemId);
 contextItem.setAttribute("select", "properties_you_need");
 contextItem = contextItem.apply();

 if (contextItem.isError())
 {
 return inn.newError("Failed to load context item");
 }

// 4. Apply business logic to determine assignees
// ... your custom logic here ...

// 5. Modify activity assignments
 Item assignment = inn.newItem("Activity Assignment", "add");
 assignment.setProperty("source_id", activityId);
 assignment.setProperty("related_id", assigneeIdentityId);
 assignment.setProperty("voting_weight", "100");
 assignment = assignment.apply();
```

```
 if (assignment.isError())
 {
 return inn.newError("Error creating assignment: " +
assignment.getErrorDetail());
 }
```

// 6. Return success

```
 return this; // Or return inn.newResult("Success message")
```

Common Code Patterns

The following examples demonstrate common patterns for implementing Workflow Assignment Rules.

Pattern 1: Assign Single Identity to Activity

Simplest pattern for assigning a specific Identity to an activity:

```
// Target = "Activity"
// this = Activity item
Innovator inn = this.getInnovator();
string activityId = this.getID();

// Define the Identity to assign (can be user Identity or group Identity)
string identityId = "IDENTITY_ID_HERE";

// Create assignment
Item assignment = inn.newItem("Activity Assignment", "add");
assignment.setProperty("source_id", activityId);
assignment.setProperty("related_id", identityId);
assignment.setProperty("voting_weight", "100");
assignment = assignment.apply();

if (assignment.isError())
{
return inn.newError("Error creating assignment: " +
assignment.getErrorDetail());
}

return this;
```

Pattern 2: Assign Manager to All Workflow Activities

Assigns the manager from the context item to all non-start/non-end activities:

```
// Target = "Workflow"
// this = Workflow Process item
Innovator inn = this.getInnovator();
string workflowId = this.getID();

// Resolve context and get manager
string contextTypeId = this.getAttribute("context_type_id");
string contextItemId = this.getAttribute("context_item_id");

Item contextItem = inn.newItem();
contextItem.setAction("get");
contextItem.setAttribute("typeId", contextTypeId);
contextItem.setID(contextItemId);
contextItem.setAttribute("select", "managed_by_id");
contextItem = contextItem.apply();

if (contextItem == null || contextItem.isError())
{
return inn.newError("Failed to load context item");
}

string managerId = contextItem.getProperty("managed_by_id");
if (string.IsNullOrEmpty(managerId))
{
return inn.newResult("No manager found; no assignments created");
}

// Query workflow activities
Item wf = inn.newItem("Workflow Process", "get");
wf.setID(workflowId);
wf.setAttribute("select", "id");
Item wfPA = wf.createRelationship("Workflow Process Activity", "get");
wfPA.setAttribute("select", "related_id");
wf = wf.apply();
```

```
if (wf == null || wf.isError())
{
return inn.newError("Failed to load workflow activities");
}
```

```
// Get activities
var activities = wf.getItemsByXPath("//Item[@type='Activity']");
if (activities == null || activities.getItemCount() == 0)
{
return inn.newResult("No activities found; no assignments created");
}
```

```
// Create assignments for non-automatic activities
int created = 0;
for (int i = 0; i < activities.getItemCount(); i++)
{
var activity = activities.getItemByIndex(i);
```

```
// Skip automatic activities
if (activity.getProperty("is_auto") == "1")
{
continue;
}
```

```
string activityId = activity.getID();
```

```
Item assignment = inn.newItem("Activity Assignment", "add");
assignment.setProperty("source_id", activityId);
assignment.setProperty("related_id", managerId);
assignment.setProperty("voting_weight", "100");
assignment = assignment.apply();
```

```
if (assignment.isError())
{
return inn.newError("Failed to create assignment for activity " +
```

```
activityId);
 }
 created++;
 }
 return inn.newResult("Created " + created + " assignments");
```

Pattern 3: Assign Part Owner Based on Affected Items

Queries affected items and assigns Part Owners to the activity:

```
// Target = "Activity"
// this = Activity item
 Innovator inn = this.getInnovator();
 string activityId = this.getID();

// Resolve context
 string contextTypeId = this.getAttribute("context_type_id");
 string contextItemId = this.getAttribute("context_item_id");

 if (string.IsNullOrEmpty(contextTypeId) ||
string.IsNullOrEmpty(contextItemId))
 {
 return inn.newError("Unable to determine context item");
 }

// Query affected items through Change Order
 Item changeOrder = inn.newItem();
 changeOrder.setAction("get");
 changeOrder.setAttribute("typeId", contextTypeId);
 changeOrder.setID(contextItemId);
 changeOrder.setAttribute("select", "id");

// Get affected items
 Item affectedRel =
```

```
changeOrder.createRelationship("ucm_ChangeOrder_AffectedItem", "get");
 affectedRel.setAttribute("select", "related_id(owned_by_id)");

 changeOrder = changeOrder.apply();

 if (changeOrder == null || changeOrder.isError())
 {
 return inn.newError("Failed to load affected items");
 }

// Collect unique part owners
 var owners = new HashSet<string>();
 var affectedItemRels =
changeOrder.getRelationships("ucm_ChangeOrder_AffectedItem");

 if (affectedItemRels != null && affectedItemRels.getItemCount() > 0)
 {
 for (int i = 0; i < affectedItemRels.getItemCount(); i++)
 {
 var rel = affectedItemRels.getItemByIndex(i);
 var affItem = rel.getRelatedItem();

 if (affItem != null && !affItem.isError())
 {
 string ownerId = affItem.getProperty("owned_by_id");
 if (!string.IsNullOrEmpty(ownerId))
 {
 owners.Add(ownerId);
 }
 }
 }
 }

 if (owners.Count == 0)
 {
```

```
return inn.newResult("No part owners found; no assignments created");
}

// Create assignment for each unique owner
int created = 0;
foreach (string ownerId in owners)
{
Item assignment = inn.newItem("Activity Assignment", "add");
assignment.setProperty("source_id", activityId);
assignment.setProperty("related_id", ownerId);
assignment.setProperty("voting_weight", "100");
assignment = assignment.apply();

if (assignment.isError())
{
return inn.newError("Failed to create assignment for owner " + ownerId);
}

created++;
}

return inn.newResult("Created " + created + " assignments for part owners");
```

Pattern 4: Query Existing Assignments Before Modification

Checks for existing assignments to avoid duplicates or to modify assignment properties:

```
// Target = "Activity"
// this = Activity item
Innovator inn = this.getInnovator();
string activityId = this.getID();

// Query existing assignments
Item existingAssignments = inn.newItem("Activity Assignment", "get");
existingAssignments.setProperty("source_id", activityId);
```

```
existingAssignments.setAttribute("select", "related_id,voting_weight");
existingAssignments = existingAssignments.apply();

// Check if specific identity is already assigned
string targetIdentityId = "IDENTITY_ID";
bool alreadyAssigned = false;

if (!existingAssignments.isError() && existingAssignments.getItemCount()
> 0)
{
for (int i = 0; i < existingAssignments.getItemCount(); i++)
{
var assignment = existingAssignments.getItemByIndex(i);
if (assignment.getProperty("related_id") == targetIdentityId)
{
alreadyAssigned = true;
break;
}
}
}

if (alreadyAssigned)
{
return inn.newResult("Identity already assigned; no action taken");
}

// Create new assignment
Item newAssignment = inn.newItem("Activity Assignment", "add");
newAssignment.setProperty("source_id", activityId);
newAssignment.setProperty("related_id", targetIdentityId);
newAssignment.setProperty("voting_weight", "100");
newAssignment = newAssignment.apply();

if (newAssignment.isError())
{
```

```
 return inn.newError("Error creating assignment: " +
newAssignment.getErrorDetail());
 }
 return this;
```

Configuring Item Types in Change Management

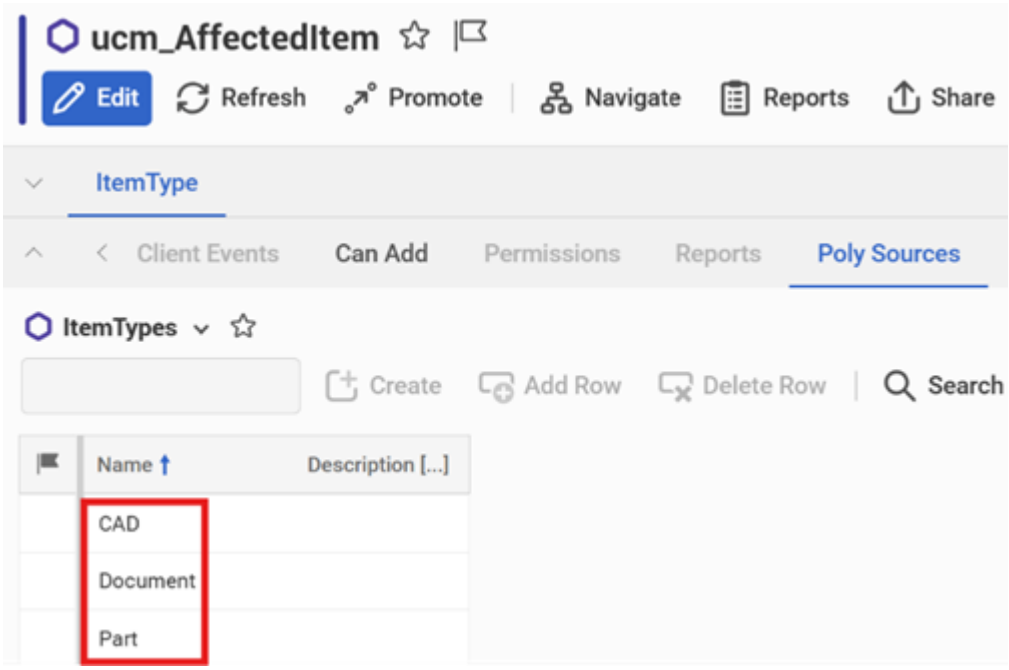
Configuring Affected Item Types for Change items

The Item Types that appear in the **Affected Items** tab of Change items are defined by the poly sources in the **ucm_AffectedItem** Item Type.

Steps to configure Poly Sources:

1. Log in as a member of the **Administrators** identity.
2. In the **Contents**, navigate to **Administration → Item Types**.
3. Search and open **ucm_AffectedItem**.
4. On the **Poly Sources** tab (toggle the tabs if necessary to locate it), update the list of Poly Source Item Types as needed.

> **Important**
>
> Each Item Type in the Affected Item poly source (ucm_AffectedItem) must include a '**Name**' property.



5. Click **Done**.

Configuring Supporting Data Item Types for Change items

The Item Types that appear in the **Supporting Data** tab of Change items are defined by the poly sources in the **ucm_SupportingData** Item Type. The steps are the same as those used to configure Affected Item Types.

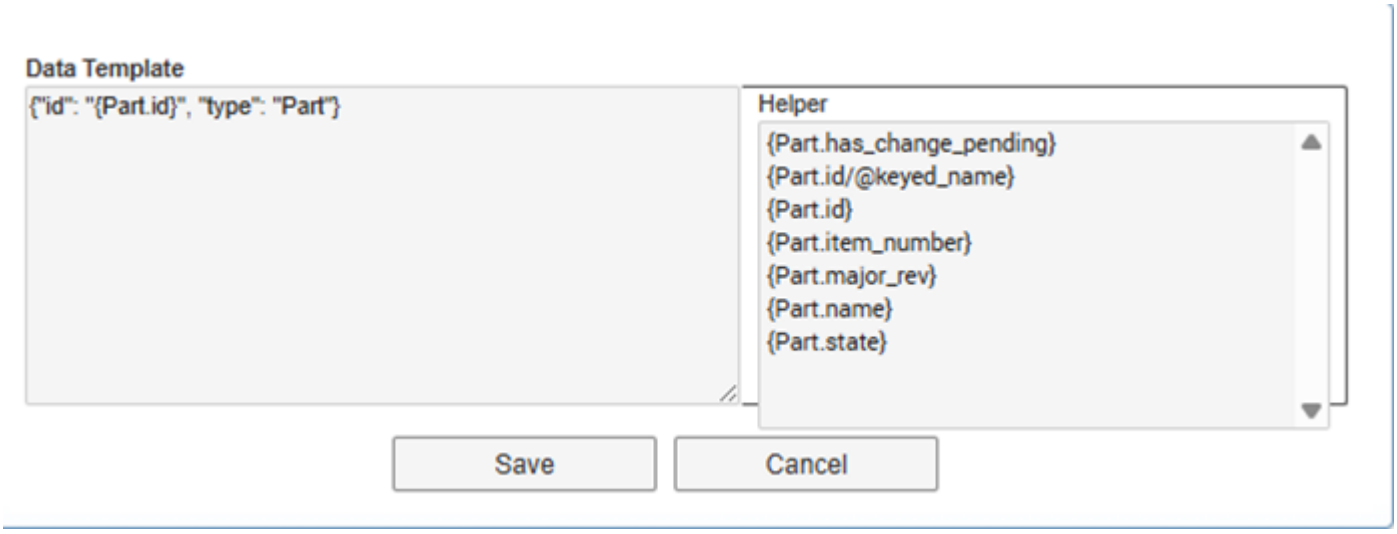
Configuring Impact Analysis Tool

The Impact Analysis Tool can be configured using Query Definitions (QD), Tree Grid Views (TGV), and Change Process Definitions (CPD). This section provides instructions for administrators on how to set up and customize impact analysis views for different item types within the Change Management system.

Configuring Impact Analysis Views

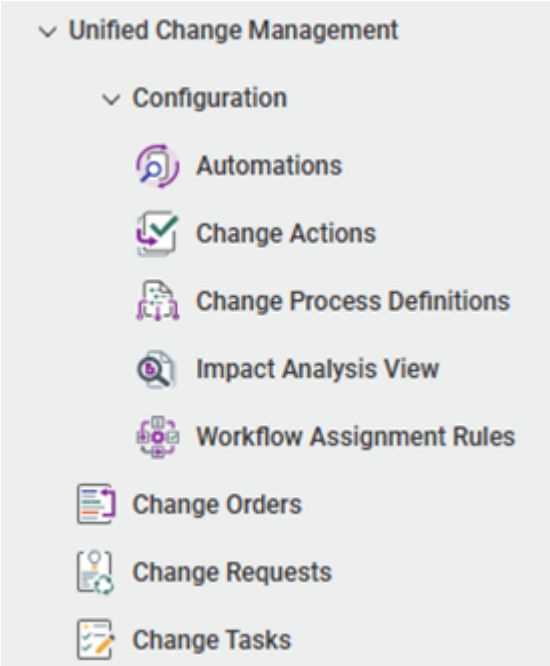
To configure an Impact Analysis View, first create the necessary Query Definitions (QD) (see QD admin guide) and Tree Grid Views (TGV) (see TGV admin guide) for specific item type, such as Part (for item types used in Change Management).

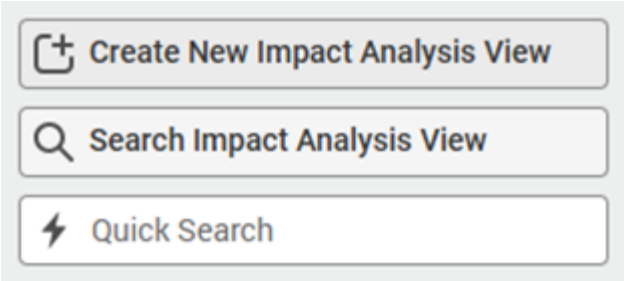
- First column of TGV should define Data Template with id, type provided (like {"id": "{CAD.id}", "type": "CAD"})



Creating Impact Analysis Views

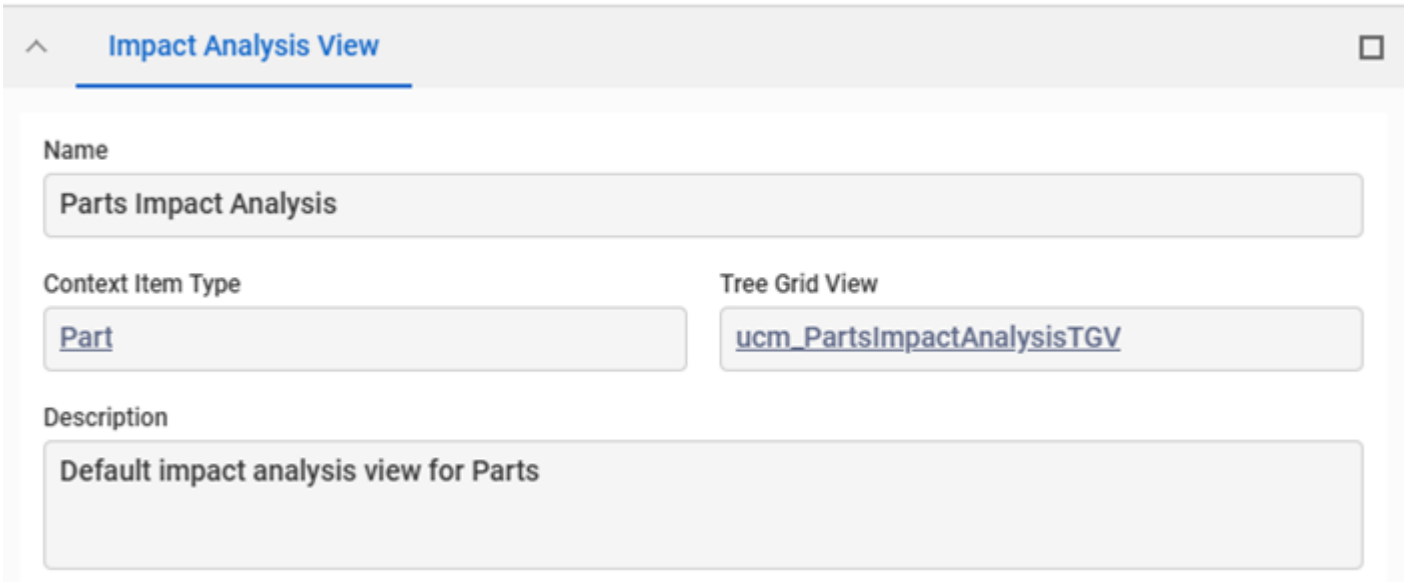
1. **Log in with Change Administrator privileges**
   - Sign in as a member of the **Change Administrators** group.
2. **Navigate Impact Analysis Views**
   - Navigate to Table of Contents → Unified Change Management → Configuration → Impact Analysis View.
   - Click **Create New** to open a new Impact Analysis View form.





3. **Fill in the required properties:**

Field	Description	Requirements
Name	Identifier for the view	Required. Maximum 128 characters. Use descriptive names that indicate the view's purpose (e.g., "Engineering Part Structure").
Context Item Type	Target Item Type	Required. Select Item for which the View is created.
Tree Grid View	TGV to be shown	Required. Select an existing TGV with the same Context Item Type.
Description	Explanation of View	Optional but recommended. Document what the view show.

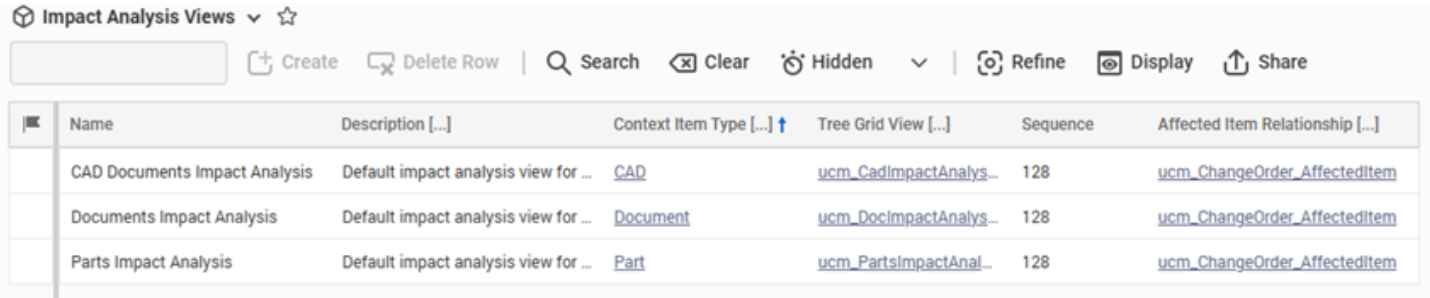


Configuring Impact Analysis View usage

Add Impact Analysis View to Change Process Definition

- **Log in with Change Administrator privileges**
  - Sign in as a member of the **Change Administrators** group.
- **Add Impact Analysis View to Change Item**
  - Navigate to Table of Contents → Unified Change Management → Configuration → Change Process Definition.
  - Open Change Process Definition, click **Edit**.
  - Go to Impact Analysis Views tab, **add** corresponding Impact Analysis View to the list
  - Fill in the required properties:

Field	Description	Requirements
Sequence	Views order, grouped by Context Item type	Required. Determines the order in which Impact Analysis Views appear in dropdown menu, and which view loads first by default.
Affected Item Relationship	Change Item relationship	Required. Specifies the target for the "Add to Change" button, defining how items are related to the change process.



Configurations for Extending Change-Controlled Items

This section describes the configurations required to extend and customize the list of change-controlled items in change management processes.

Configuring Change Orders to Release Process Plans

In this example, the business uses Change Orders to **release** MBOMs and their associated work instructions, represented as Process Plan items in Aras Manufacturing Process Planning (MPP).

Steps to configure are as follows:

1. **Log in with Administrator privileges**
   - Sign in as a member of the **Administrators** group.
2. **Update Poly Sources for Affected Items**
   - Open the **ucm_AffectedItem** Item Type for editing.
   - Go to the **Poly Sources** tab and add the **mpp_ProcessPlan** Item Type.
3. **Log in with Change Administrator privileges**
   - Sign in as a member of the **Change Administrators** group.
4. **Update the Change Actions**
   - Open the **Release** Change Action.
   - On the **Applicable To** tab, add the **mpp_ProcessPlan** Item Type.
   - Repeat for the **Revise** Change Action and any other relevant Change Actions.
5. **Edit the Change Process Definition to add validation automations**
   - **Purpose:** Ensure a validation error appears if affected items of type **Process Plan** with the **Release** action are not in the **Preliminary** state.
   - While logged in as **Change Administrators**, open the **Change Process Definition** for the **Change Order** Item Type.
   - On the **Automation Rules** tab, add the existing automation named *Selected action requires item to be in unreleased state*, and fill in the fields as required.

Field		
Enabled	✔ Checked	✔ Checked
Event	Add	Update
Automate For / (Applicable To)	Relationship	Relationship
From State / To State	[empty]	[empty]
Relationship	ucm_ChangeOrder_AffectedItem	ucm_ChangeOrder_AffectedItem
Related Item Type	mpp_ProcessPlan	mpp_ProcessPlan
Related Event	Release	Release

   - Repeat the above steps to trigger a validation error if affected items of type Process Plans with the Revise action are not in Released state. Add the *Selected action requires item to be in released state* automation and fill the fields as follows:

Field		
Enabled	✔ Checked	✔ Checked
Event	Add	Update
Automate For / (Applicable To)	Relationship	Relationship
From State / To State	[empty]	[empty]
Relationship	ucm_ChangeOrder_AffectedItem	ucm_ChangeOrder_AffectedItem
Related Item Type	mpp_ProcessPlan	mpp_ProcessPlan
Related Event	Revise	Revise

6. **Edit the Change Process Definition to add action automations**
   - Purpose: Change Orders with affected Process Plans should automatically promote the Process Plans throughout the Change Order workflow, including (but not limited to):
     - Promote to **In Review** when the workflow reaches the **Audit** activity.
     - Promote from **In Review** back to **Preliminary** if the workflow is returned for rework.
     - Promote to **Released** when the Change Order workflow is completed.

Purpose	Automation	Method
New revision of affected item.	Create a new revision	ucm_CreateNewRevision
Promote affected item to In Review.	Promote new item revision to In Review for Audit	ucm_PromoteNewItemRevForAudit
Promote affected item from In Review back to Preliminary.	Return new item revision to Preliminary state	ucm_PromoteNewItemRevForRework
Promote affected item to Released.	Release item	ucm_ReleaseItem

   - The **same steps** can apply to other action automations as necessary, e.g. obsoleting items.

Important

The methods for Release, Revise, and Obsolete actions do not require modification, as they are implemented generically for any Item Type. If different business logic is needed, a new method can be created.

Important

Process Plans reuse existing **Change Actions** and lifecycle state names. If your setup uses different actions, states, or logic, update the method or create a new automation.

7. Perform additional checks
    - Ensure the Item Type configuration supports the proper behavior for change-controlled items. For example, Process Plans should be versioned through Change Orders, and once released, they should no longer be editable.
        - **Confirm** the **mpp_ProcessPlan** Item Type is versionable.
        - **Verify** the **Released** state prevents edits (using state permissions or the *not lockable* setting). This ensures users can't modify released Process Plans (operations, steps, tools, produced/consumed parts, etc.).
        - **Review** any other relevant Item Type configurations.
        - In the lifecycle map, check which role promotes lifecycle transitions (for Process Plans, this is usually **Manufacturing Engineering**) and confirm **Change Executor** is included in that role identity.

Configuring Change Requests to Plan Change of Process Plans

In this example, the business uses Change Requests to **initiate change** to MBOMs and their associated work instructions, represented as Process Plan items in Aras Manufacturing Process Planning (MPP).

Steps to configure are as follows:

1. **Follow** steps 1 to 4 from **8.1 Configuring Change Orders to Release Process Plans**, if they haven't been already executed.
2. **Edit** the Change Process Definition to add validation automations.
   - **Purpose**: Ensure a validation error appears if affected items of type Process Plan with the Release action are not in the Preliminary state.
   - While logged in as **Change Administrators**, open the Change Process Definition for the Change Request Item Type.
   - On the Automation Rules tab, add the existing automation named *Selected action requires item to be in unreleased state*, and fill in the fields as required.

Field		
Enabled	✔ Checked	✔ Checked
Event	Add	Update
Automate For / (Applicable To)	Relationship	Relationship
From State / To State	[empty]	[empty]
Relationship	ucm_ChangeRequest_AffectedItem	ucm_ChangeRequest_AffectedItem
Related Item Type	mpp_ProcessPlan	mpp_ProcessPlan
Related Event	Release	Release

   - **Repeat** the above steps to trigger a validation error if affected items of type Process Plans with the Revise action are not in Released state. Add the *Selected action requires item to be in released state* automation and fill the settings as follows:

Field		
Enabled	✔ Checked	✔ Checked
Event	Add	Update
Automate For / (Applicable To)	Relationship	Relationship
From State / To State	[empty]	[empty]
Relationship	ucm_ChangeRequest_AffectedItem	ucm_ChangeRequest_AffectedItem

Related Item Type	mpp_ProcessPlan	mpp_ProcessPlan
Related Event	Revise	Revise

Configuring Change Tasks to Perform Changes in Process Plans

Change Tasks track updates and approvals to the MBOM and work instructions for a Process Plan. As they are part of a Change Order and pre-configured in **8.1 Configuring Change Orders to Release Process Plans**, no further configuration is needed to use Process Plan items in Change Tasks.

Resulting Item description

Resulting Item is a property on both **ucm_ChangeTask_AffectedItem** and **ucm_ChangeOrder_AffectedItem** relationship items. It points to the specific item that represents the outcome of the change.

When you work with Affected Items in a Change Task or Change Order, the **Resulting Item** can take one of three values by default:

- If the Change Item is no longer active: **Actioned Item**.
- If Change Item is active and there andan Actioned Item already exists: The system shows the latest version of that Actioned Item.
- **If neither of the above applies**: latest version of an **Affected Item.**

You can configure how the **Resulting Item** behaves, but you cannot edit this property directly because its value is calculated automatically. If you need to change it, you must set an **Actioned Item** on the *ucm_ChangeTask_AffectedItem* or *ucm_ChangeOrder_AffectedItem* relationship.

By default, only the **Change Executor** can create or update an Actioned Item. If you want a user to make this change manually, you need to create an **ucm_Automation Action** that sets the Actioned Item for the relationship item.

Important

Do not change Resulting Item directly, work with Actioned Items (see ucm_Automation with name *Set Actioned Items* in Change Task Change Process Definition as the reference).

Important

If you want to customize how the **Resulting Item** behaves without creating conflicts, you need to either disable or update the ucm_Automation named *Set Actioned Items* in the Change Task Change Process Definition

Configuring UCM and Legacy Change Management Compatibility

Unified Change Management (UCM) and legacy Product Engineering (PE) Change Management can run together in the same environment. Compatibility rules ensure that any item being changed is controlled by only one system at a time.

Compatibility Rules

- **PE items cannot be promoted or revised manually** if the item is currently managed (or was previously managed) by a Change Order, except for items in Canceled Change Orders.
  - Enforced by server methods: ucm_BlockVersionIfInUCMChange and ucm_BlockPromoteIfInUCMChange
  - These methods are configured as onBeforeVersion and onBeforePromote events on all Item Types in the ucm_AffectedItem polysource.
  - The method ucm_SkipUCMValidationOnPEChange is also configured on the same events and must execute before the blocking methods to allow authorized PE Change Items to bypass the validation.
- **Items cannot be added to a Change Order** if they are included in an active PE Change Item (is_released = 1). The following PE Change Item types are checked:
  - Express DCO
  - Express ECO
  - ECN
  - Simple ECO
  - Express EDR
- **Legacy Change Management items can promote and revise PE Affected Items** that were previously managed by UCM. The following PE Change Item types are allowed to bypass UCM validation:
  - Express DCO
  - Express ECO
  - ECN
  - Simple ECO
  - ECR

Pre-Configured Validation

Compatibility is enforced through a pre-configured validation automation named **"Affected Items are not included in any active PE Change Items"**. This validation:

- Prevents affected items from being added to a Change Order if they are part of an active PE Change Item.
- Uses the method ucm_NotInAnyActivePEChange to perform the validation.
- Can be configured within the Change Process Definition for Change Orders.

Server Method Configuration

Automatic guardrails are applied through server methods:

Method Name	Event	Purpose	Execution Order
ucm_SkipUCMValidationOnPEChange	onBeforeVersion, onBeforePromote	Allows authorized PE changes to proceed	First
ucm_BlockVersionIfInUCMChange	onBeforeVersion	Prevents manual revision of items during active Change Orders	Second
ucm_BlockPromoteIfInUCMChange	onBeforePromote	Prevents manual promotion of items during active Change Orders	Second

These methods are applied automatically to all Item Types in the ucm_AffectedItem polysource to ensure consistent enforcement across all change-controlled items.

Important

The execution order is **critical**. The ucm_SkipUCMValidationOnPEChange method must be configured to execute before the guardrail methods to allow authorized PE Change Items to proceed with promotion and revision operations.

Customizing PE Change Item Types

To modify the list of PE Change Item types that are allowed to bypass UCM guardrails:

1. **Log in with Administrator privileges**
    - Sign in as a member of the Administrators group.
2. **Locate the bypass method**
    - Navigate to Administration > Methods
    - Search for and open the method ucm_SkipUCMValidationOnPEChange
3. **Update the authorized PE Change Item Types list**
    - In the method code, locate the list of authorized PE Change Item types.
    - Add or remove item types as needed for your organization's requirements.
    - Ensure only trusted PE Change Item types are included in this list.

To change which PE Change Item types the system checks for conflicts when you add items to a Change Order, you can update the list as needed:

1. **Locate the validation method**
    - Navigate to Administration > Methods
    - Search for and open the method ucm_NotInAnyActivePEChange
2. **Update the PE Change Item Types list**
    - In the method code, locate the PEChangeItemTypes list.
    - Add or remove item types as needed for your organization's requirements.
    - Save the changes and test the validation with your specific PE Change Item types.

Product Engineering: Changes Pending Flag

The *has_change_pending* checkbox is part of the Product Engineering (PE) application and indicates whether an item (Part, CAD Document, or Document) is currently in an active PE change.

- If the item is part of an active PE Change → the checkbox is checked (True)
- If the item is NOT part of an active PE Change → the checkbox is unchecked (False)

This field was designed only for PE workflows.

Behavior When Used with Unified Change Management

With Unified Change Management, when a Change Order reaches the **In Work** state, the *has_change_pending* checkbox is set to true for all affected items. However**,** if the item is later revised (either manually or through another process), the PE method (***PE_update_has_change_pending***) will automatically clear the checkbox. This behavior is expected and does not impact UCM processes. All active UCM Change Order statuses are visible in the item's **Changes** tab. The checkbox is available for PE compatibility only.

Best Practices

Recommendations:

- Use UCM relationships (e.g. ucm_ChangeOrder_AffectedItem, ucm_ChangeTask_AffectedItem) for reporting to provide accurate UCM status.
- Use the **Changes** tab on Affected Items to view active UCM Change Orders.

For customers with existing PE customizations:

- UCM does not modify the PE_update_has_change_pending method; your existing PE logic continues to work as designed.
- UCM leverages separate validation methods (ucm_BlockVersionIfInUCMChange and ucm_BlockPromoteIfInUCMChange) to prevent users from manually creating new versions or promoting lifecycle states of items that are in active Change Orders. These validations ensure change control integrity.

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.

Installation Guide

Introduction

Purpose

The Installation Guide describes how to install the Release 30 of Aras Unified Change Management.

Scope

The Installation Guide explains the installation process for installing Unified Change Management for Aras Innovator. The installation involves updates to both the code tree and database.

Target Audience

This Installation Guide is intended for Aras Innovator administrators.

Prerequisites

To install Aras Unified Change Management, users need to confirm that the installation of Aras Innovator meets the minimum requirements to run it.

Checking for Eligibility

The following steps outline the process to verify the code tree and database are eligible for installing Aras Unified Change Management application:

1. Log into **Aras Innovator** as an administrator.
2. Click the **User** icon in the top-right corner and select **About**.
3. Confirm that the dialog shows the following:
    - **Release 30** (or higher).
4. Close the dialog.
5. Navigate to the main tree view. Go to Administration → Variables.
6. Confirm that one of following sets of variables are listed:

VersionMajor	VersionMinor	VersionServiceUpdate	VersionLabel
14	0	22	Release 30
14	0	25	Release 31
14	0	28	Release 32
14	30	0	Release 33
14	34	0	Release 34
14	35	0	Release 35
14	36	0	Release 36
14	37	0	Release 37

If it does not meet the service pack requirements listed in the previous steps, contact Aras to discuss options.

Prerequisites for Aras Innovator Release 30 and Release 31

Before installing Unified Change Management Release 30 on Aras Innovator Release 30 and Aras Innovator Release 31, the Aras Innovator .NET 8 **hotfix must be applied**. The hotfixes are available for subscribers here:

- Innovator Release 30
- Innovator Release 31

Installing Aras Unified Change Management

Aras Unified Change Management must modify the Aras Innovator database with the metadata required to run the application. Before applying these changes, verify the following requirements:

Requirements:

- Aras Innovator Release 30 (or higher)
- Aras Update Tool 1.23 (or higher)

Notification and Backup

1. Notify users that the system will be down at a scheduled time, and they should log out of the system prior to the start of the process.

   Important

   It is best to give at least 24-hour notice, as well as a reminder 15 minutes prior to the upgrade.

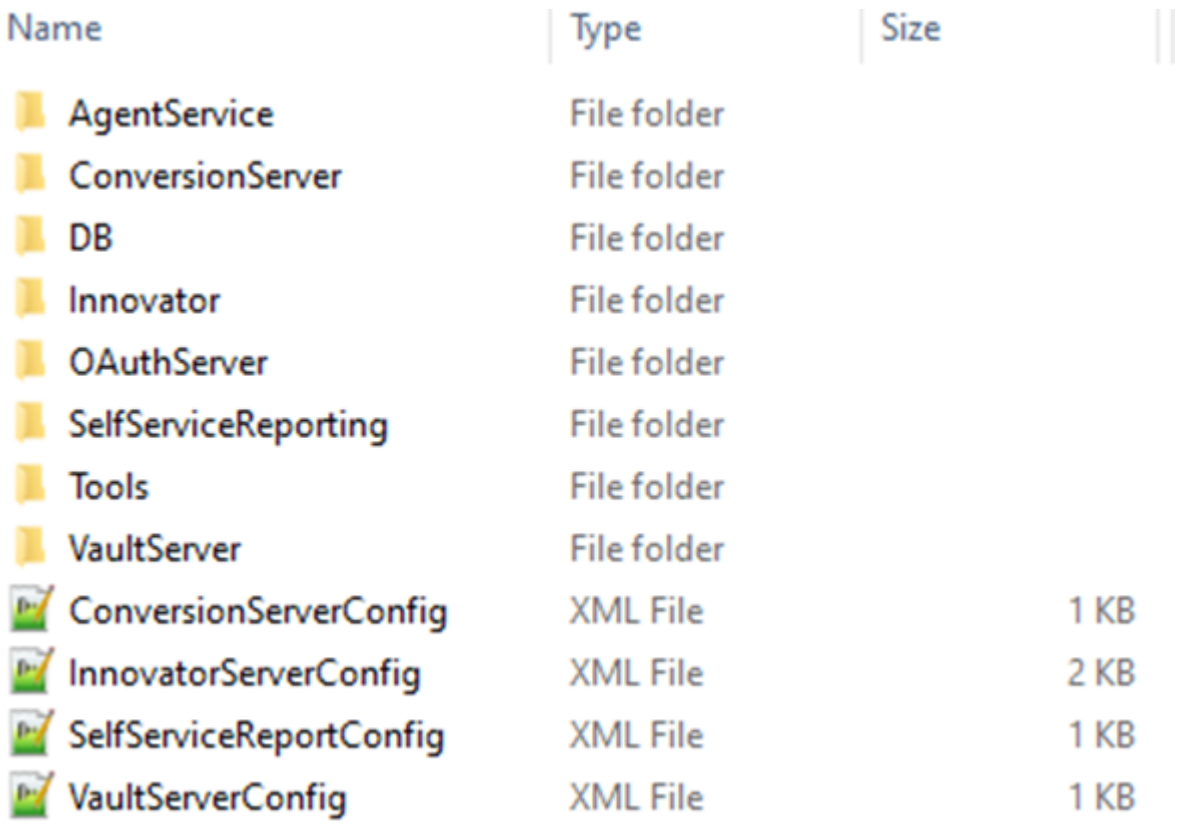
2. Backup the code tree.

   The “Code Tree” refers to files and folders installed to the disk when Aras Innovator was first installed.

   The default path for the Code Tree installation would be something like:

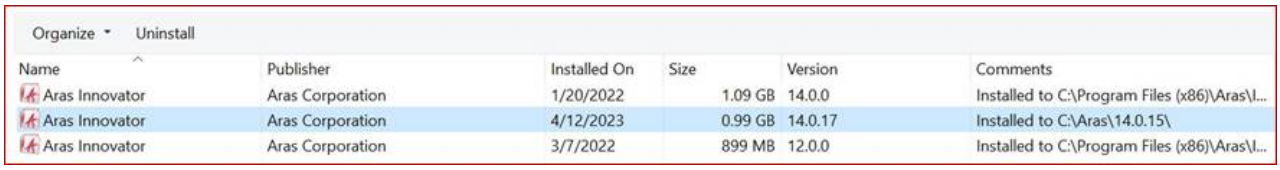
   ```
   C:\Program Files (x86)\Aras\Innovator
   ```

   With the following contents:



   The following steps can be used to confirm the installation folder:

1. From the Windows search field, enter **Control** to access the **Control Panel**
2. Select **Programs and Features**
3. Right-click the **Name** column and select **More…**
    - Select the Comments header if it is not checked
    - Click OK
4. Search for the **Aras Innovator** entry in the **Programs and Features** window.
5. View the value set in the **Comments** column for the Aras Innovator entry:



After the tree code installation path has been verified, back up the folder and all its contents.

3. Disconnect all users from the database.

   The easiest way to prevent client sessions from committing any further changes to the database is to change the database connection string in the `InnovatorServerConfig.xml` from `<DB-Connection` … to `<xDB-Connection` and restart the w3svc service (IIS). This expires all session and prevents all new connections to the Aras Innovator database through the existing instance.

4. Backup the database.
5. Place these files in a safe location, as they will be needed to restore if the process fails.

6. Enable database connections.

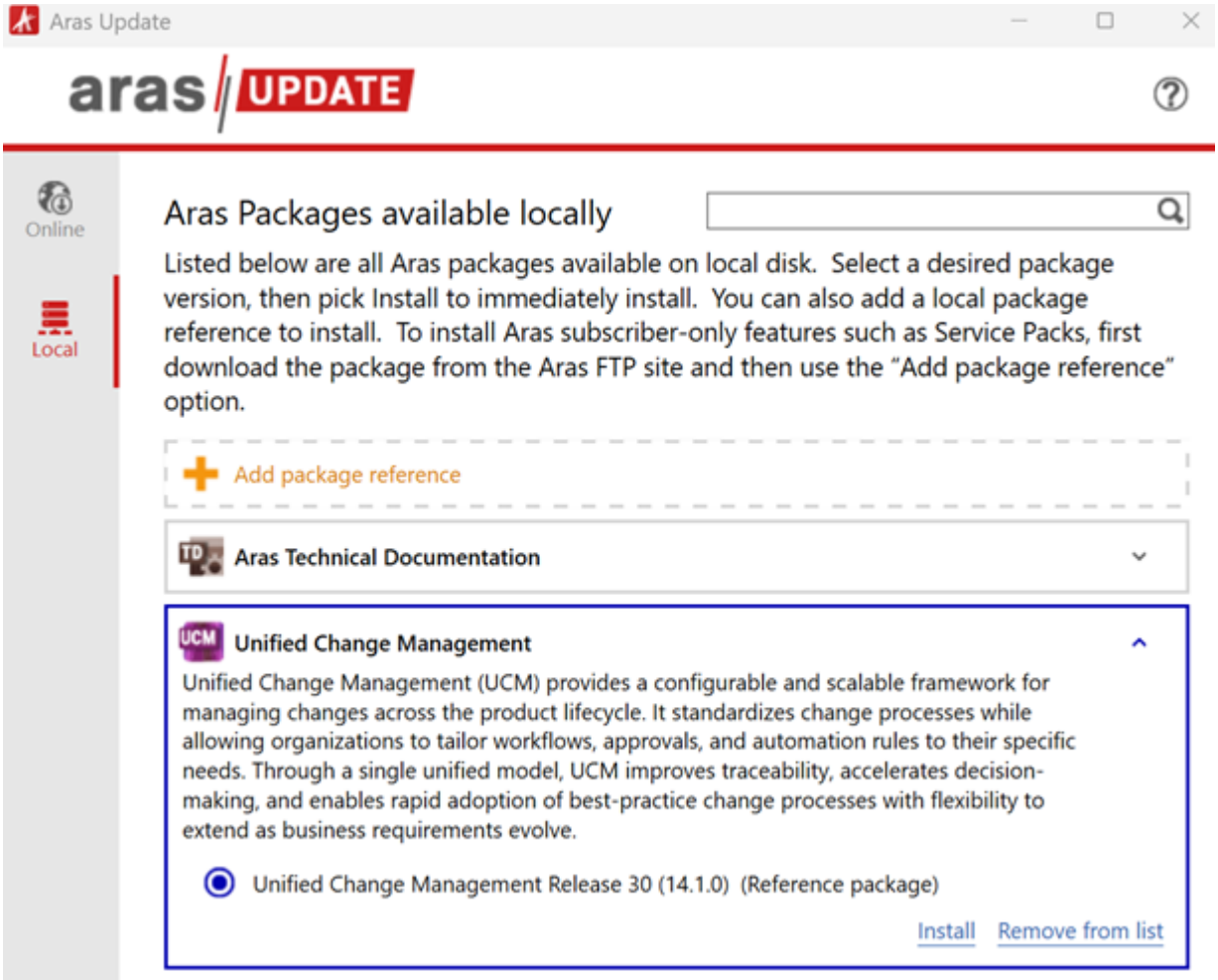
   After the database backup has been completed, enable the database connection string in the `InnovatorServerConfig.xml` by changing it from `<xDB-Connection` … to `<DB-Connection` and restarting the w3svc service (IIS).

Installing Aras Unified Change Management application

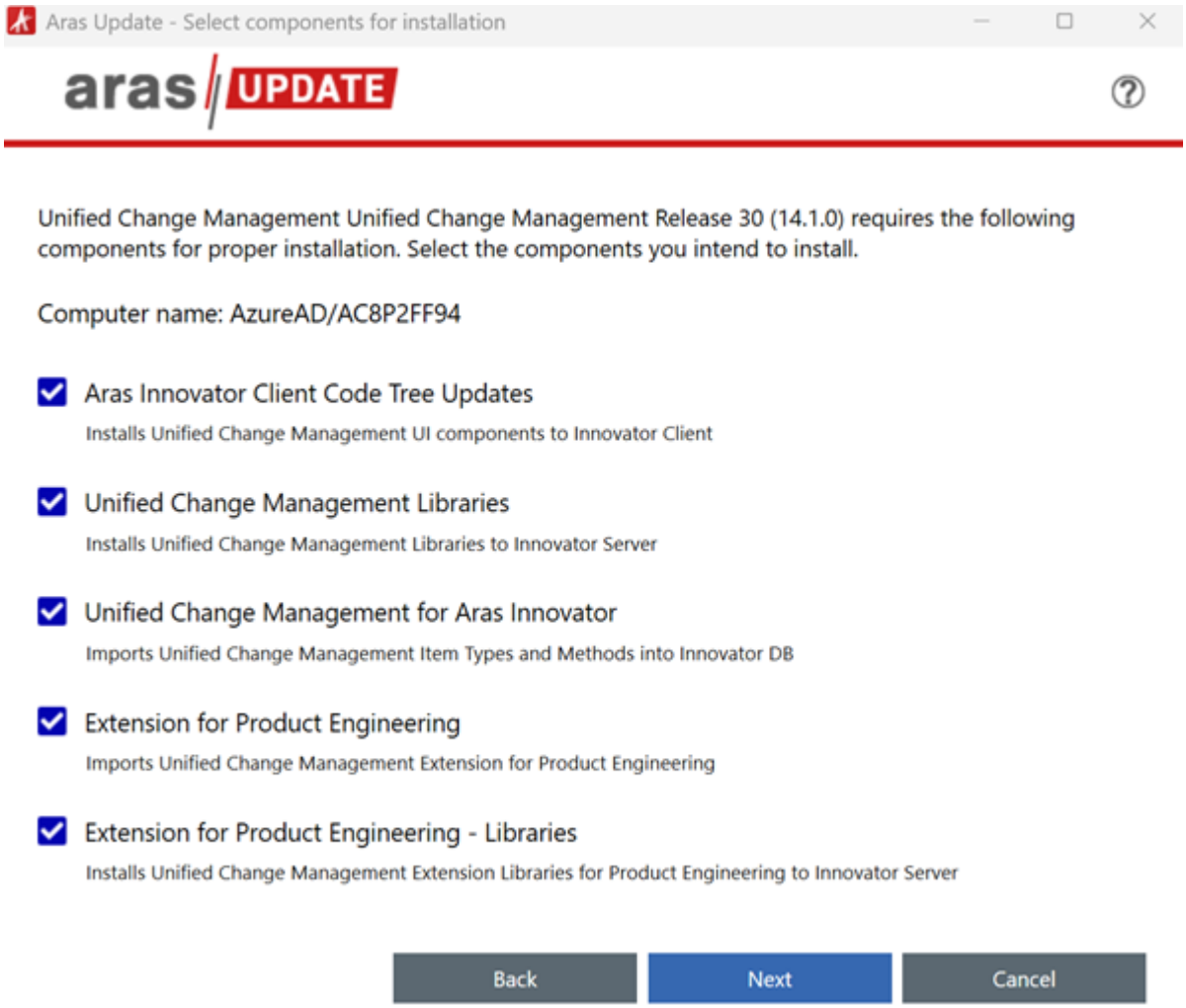
Aras Unified Change Management application can be installed using the Automated Installation option (using the Aras Update tool).

Automated Installation Option

1. Copy and unzip the Aras Unified Change Management CD Image on the local computer.
2. Enable the **Super User** login.
3. Launch the `ArasUpdate.exe` file as administrator.
4. On a default installation, the `ArasUpdate.exe` file can be found in the `C:\Program Files (x86)\Aras\Aras Update\ directory` .
5. Locate and select the **Aras Unified Change Management** package.



6. Click **Install**.

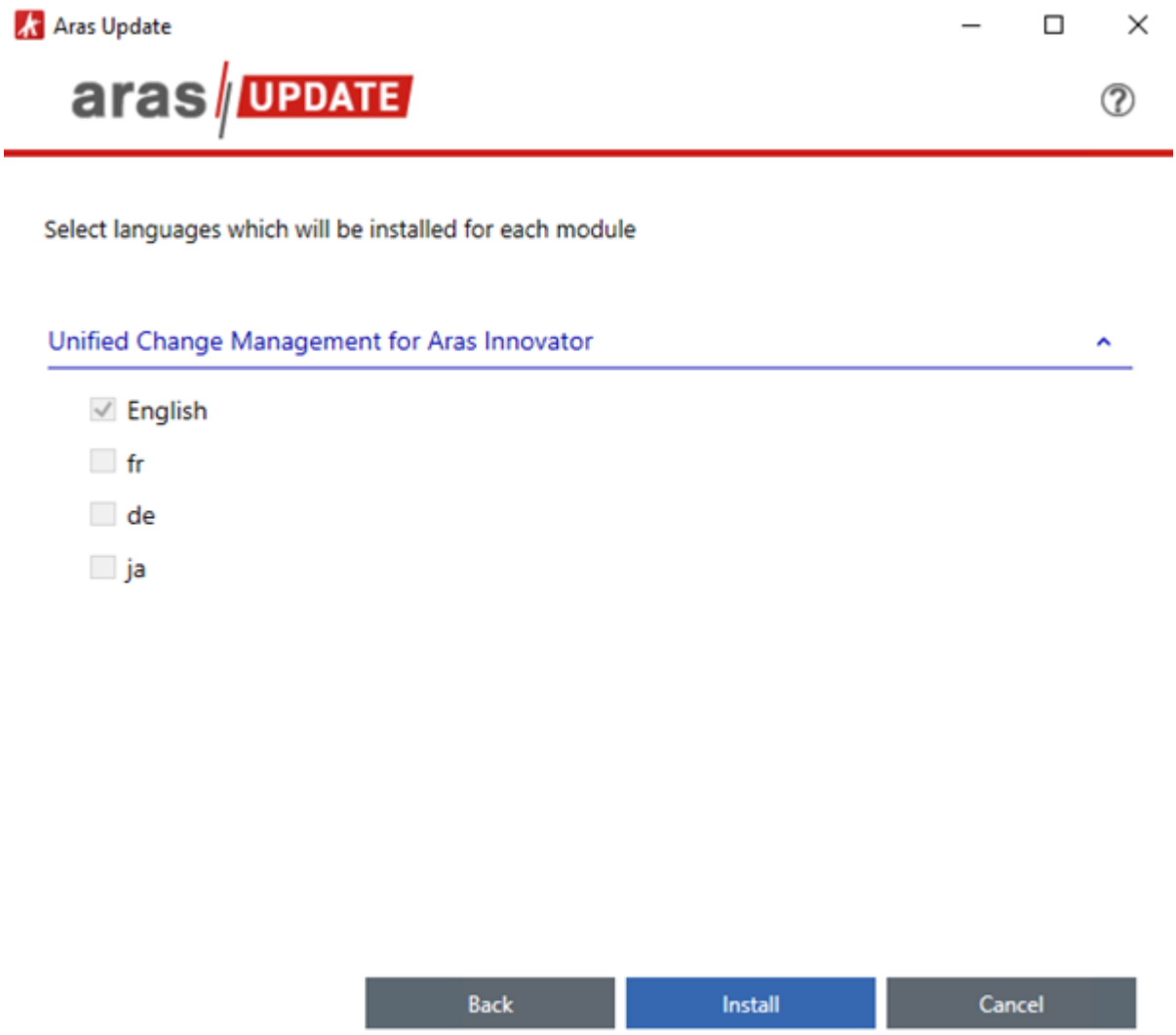


7. Select the **Aras Innovator Client Code Tree Updates**, **Unified Change Management Libraries** and **Unified Change Management for Aras Innovator** options.
8. (Optional) Select the **Extension for PE** option.
9. Click **Next**.
10. Select the appropriate logging option and click **Next**.
    The logging option is used to record the installation attempt and can be used to troubleshoot issues.
11. Input the connection information and click **Next**.
    - **Innovator Client directory** = Path to Innovator Client folder on the server (C:\ARAS\"instance name"\Innovator\Client)
    - **Innovator Server directory =** Path to Innovator Server folder on the server (C:\ARAS\"instance name"\Innovator\Server)
    - Server URL = The connection URL for Aras Innovator
    - Database = The target Aras Innovator database

- Username = root
- Password = Password for “root” login (Default is “innovator”)

Aras Update should import everything required for Aras Unified Change Management.

12. Select the appropriate languages to install and click **Install**.



13. After the import is complete, disable the Super User login.

Confirming the Installation

Use the following procedure to check if your database has been updated correctly:

1. Log in to Aras Innovator as an administrator.
2. From the TOC, select **Administration** → **Variables**.
3. Confirm the following sets of variables are listed:

Unified Change Management
14.1.0

If at any time the installation fails, revert to your backups and contact Aras Support at support@aras.com .

Post Installation

It is recommended that you review the *Aras Unified Change Management – Administration Guide* after the installation is complete.

For end users, the *Aras Unified Change Management – User Guide* is available in the Documentation folder of the CD Image.

Installation Guide - SaaS

Introduction

Purpose

Aras Unified Change Management – Installation guide for Aras DevOps provides information for Aras DevOps and Enterprise subscribers to add Release 30 of Aras Unified Change Management to their Aras Innovator instance.

Scope

This document provides instructions for installing the Beta version of Aras Unified Change Management which includes steps related to import.mf config changes.

Target Audience

This document is intended for Aras DevOps users responsible for installing and configuring the applications and platform components for the Aras Innovator instance utilizing SaaS.

Prerequisites

Aras DevOps administrators or contributors must have an installed Aras Innovator prior to following the steps outlined in this document. It is recommended that all administrators read the Aras DevOps – User Guide, as it explains how to effectively utilize the Aras DevOps service included as part of the Aras Enterprise subscription to manage Aras Innovator customizations throughout the entire lifecycle of the Aras Innovator implementation.

Definitions

This section defines the terms used in this document.

Term	Definition
Code Tree patch	The set of files in an application installation package that should be applied to the code tree to install the code tree component of an application.
Import package	The set of XML files and the corresponding import.mf file that represent a package or set of packages that need to be installed to apply the database changes of an application.
Config transformation	The transformation to be sourced into project repository corresponding to changes in configuration files of Aras Innovator code tree.

Installing Aras Unified Change Management

Retrieve Application package

1. Download the **Aras Unified Change Management CD Image** from aras.files.com site.
2. Unzip the **Aras Unified Change Management CD Image** on the local machine.

Include application Code Tree Patch into the repository

1. Navigate to the (unzipped) Aras Unified Change Management folder on the local machine.
2. Copy Innovator folder form the unzipped Aras Unified Change Management and paste it into CodeTree folder in Work.git repository.
3. Navigate to …\CodeTree\Innovator\Server\bin folder of the Work.git repository and create directory \applications\ucm
4. Move

```
Aras.UnifiedChangeManagement.dll,
Aras.UnifiedChangeManagement.Abstractions.dll,
shared.txt
from
…\Innovator\Server\bin
to
…\CodeTree\Innovator\Server\bin\applications\ucm
```

Important

DLL and .txt file are placed inside …\CodeTree\Innovator\Server\bin\applications\ucm.

Include application package into the repository

1. Copy the **unifiedchangemanagement, unifiedchangemanagement-core, PlmUnifiedChangeManagementExtension** folders from the (unzipped) ..\Imports\.. folder into the AML-packages folder of the Work.git repository.
2. Copy the **PLM** folder from the (unzipped) ..\Imports\.. folder into the AML-packages folder of the Work.git repository in case it doesn't contain it. In case it contains, copy only files which are inside the **PLM** folder.

> **Important**
>
> Do not copy the **imports.mf** file, as it will overwrite the existing **imports.mf** file in the Work.git repository.

3. Open the **imports.mf** file ..\Imports\imports.mf.
4. Copy the following <package> elements:

```
<imports>
  <package name="com.aras.innovator.solution.PLM" path="PLM" />
  <package name="com.aras.innovator.solution.unifiedchangemanagement-core" path="unifiedchangemanage
    <dependson name="com.aras.innovator.solution.PLM" />
  </package>
  <package name="com.aras.innovator.solution.unifiedchangemanagement" path="unifiedchangemanagement"
    <dependson name="com.aras.innovator.solution.unifiedchangemanagement-core" />
  </package>
</imports>
```

5. Paste the <package> elements into the **imports.mf** file of the Work.git repository.
6. Open the **imports.mf** file ..\ `PlmUnifiedChangeManagementExtension\imports.mf.`
7. Copy the following <package> elements:

```
<imports>
<package
name="com.aras.innovator.solution.PlmUnifiedChangeManagementExtension"
path="Import">
<dependson name="com.aras.innovator.solution.PLM" />
<dependson name="com.aras.innovator.solution.unifiedchangemanagement"
/>
</package>
</imports>
```

8. Paste the <package> elements into the **imports.mf** file of the Work.git repository.
9. Change the next element in the resulting config:

```
<package name="com.aras.innovator.solution.PlmUnifiedChangeManagementExtension"
path="Import">
```

to

```
<package name="com.aras.innovator.solution.PlmUnifiedChangeManagementExtension"
path="PlmUnifiedChangeManagementExtension\Import">
```

10. Save and close **imports.mf** file of the Work.git repository.

Include Config Transformation

1. Add a Configuration transformation file for the *method-config.xml* in the ..\TransformationsOfConfigFiles\Innovator\Server folder of the Work git repository with the following content:

```
<MethodConfig xmlns:xdt="<bsp-code-inline>http://schemas.microsoft.com/XML-Document-Transform</bsp-code-inline>">
<ReferencedAssemblies>
<name xdt:Transform="Insert">$(binpath)/applications/ucm/Aras.UnifiedChangeManagement.Abstractions.dll</nam
</ReferencedAssemblies>
</MethodConfig>
```

Test and Deploy Changes

1. Stage and review the changes to ensure that no customizations are overwritten.
2. Commit the changes.
3. Run the “**continuous-integration**” pipeline.
4. Run the “**deploy-innovator**” pipeline to complete the installation.

User Guide

Introduction

Purpose

This guide explains how to use **Unified Change Management** to manage updates to change-controlled items—release, revision, and obsolescence—throughout the product lifecycle. It showcases a unified approach that brings together **Change Requests** and **Change Orders** to streamline identification, approval, implementation, and validation, with seamless carry-forward of data across the process.

It demonstrates how configurable change workflows, broad item-type support, enhanced visibility (assignments and status), and user-friendly impact analysis can be tailored to your organization's processes.

Scope

You can manage changes through:

- **Change Requests and Change Orders** – Streamlined management of solution identification, approval, implementation, and validation with seamless data carry-forward.
- **Change Tasks** – Trackable units of work within a Change Order's full scope, assigned to specific users, groups, or teams to ensure accountability and progress tracking.
- **Configurable Workflows** – Customizable workflow activities, validations, transitions, and automations tailored to customer processes.
- **Broad Item Type Support** – Provides change control across diverse items within Aras applications.
  Planned for the general availability product release:
  - **New Impact Analysis** – Available on both Change Requests and Change Orders, it shows data related to the changed item, helping users determine which associated data may be impacted and need to be updated.

Target Audience

This guide is intended for users who propose, review, approve, and implement product and process changes using Change Requests and Change Orders.

Basic familiarity with Parts, CAD Documents, Documents, lifecycles, and revisions is assumed.

Introduction to Unified Change Management

Unified Change Management overview

Unified Change Management (UCM) standardizes the identification, evaluation, approval, and execution of changes across the product lifecycle—from analyzing impact to approving a solution and implementing the change. UCM unifies Change Requests and Change Orders, ensuring that data and decisions carry forward so updates to change-controlled items—such as Parts, CAD Documents, and Documents—are executed in a controlled, traceable, and efficient manner. The system comes pre-configured but can be tailored to meet specific business requirements.

Key Features:

- **Change Process Coverage** – Supports the full lifecycle of a change, including creation, review, approval, implementation, and closure, along with item release, revision, and obsolescence, with streamlined execution through Change Requests and Change Orders.
- **Affected Items and Impact Analysis** – Captures items to be changed and presents related items that may also need updates based on where-used, where-referenced, and property-level impacts.
- **Supporting Data and Files** – Maintains related business data and attachments relevant to the change.
- **Collaboration and Approvals** – Enables team collaboration with defined roles, responsibilities, and signoff tracking.
- **Centralized Change Control** – Manages multiple item types within a single environment, with configurable change-controlled item types.
- **Flexible Configuration and Efficiency** – Tailors workflows, validations, and transitions to business needs while reducing manual effort and accelerating change execution.

Value Proposition of Unified Change Management

- Ensures traceability and compliance with industry standards.
- Reduces errors and delays in implementing changes.
- Enhances decision-making with real-time visibility into change status.
- Increases efficiency through automated, standardized workflows and configurable impact analysis.
- Provides low-code/no-code configurability and adaptable UI components for managing changes across Aras applications.
- Breaks changes into trackable tasks within a Change Order, assigned to users, groups, or teams for accountability.

Change Requests

Overview

A **Change Request** is a process initiated to propose and evaluate modifications to a product to ensure that any changes are well-documented, justified, and reviewed to avoid unintended consequences.

Purpose of Change Requests

A Change Request is used to formally initiate, document, and evaluate the need for a product or process change. It serves as the entry point to change management, ensuring that all proposed modifications are reviewed, justified, and aligned with business and technical objectives before implementation. The following are common use cases for initiating a Change Request.

- **Identify issues or opportunities**
- **Field problems** – product defects or safety concerns
- **Customer or regulatory requirements** – changes driven by feedback or compliance
- **Improvement opportunities** – cost reduction, performance, or manufacturability enhancements

Key Roles in the Change Request Process

- **Change Requestors** - Initiate a change by submitting a request with all necessary supporting details, data, and justification. They define the initial scope and rationale for the change.
- **Team Members** - Contribute additional information, help identify affected items, and propose specific actions to implement the change.
- **Change Managers** - Monitor and oversee the progress of change requests, ensure proper coordination across teams, update requests as permitted by their access level, and handle escalated situations.
- **Change Review Board** - Evaluate proposed changes when requested, providing approval or rejection (go/no-go) based on impact, feasibility, and alignment with organizational priorities.
- **Change Viewers** - Access and review change requests to stay informed, without the ability to modify or approve changes.
- **Change Administrators** - Configure and maintain the change management system, support users in case of errors, and ensure smooth operation of the process.

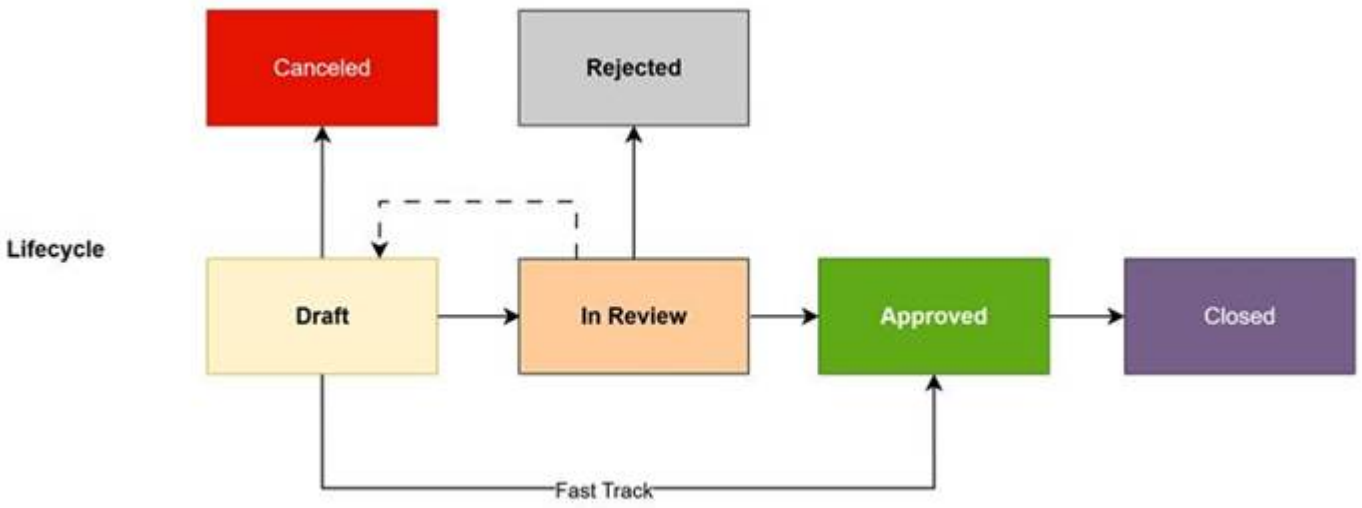
Important

Role access depends on Change Request lifecycle state; refer to the *Aras Unified Change Management - Administrator Guide*.

Change Request Lifecycle Overview

Change Requests progress through various lifecycle stages, which are primarily managed automatically as the workflow advances:

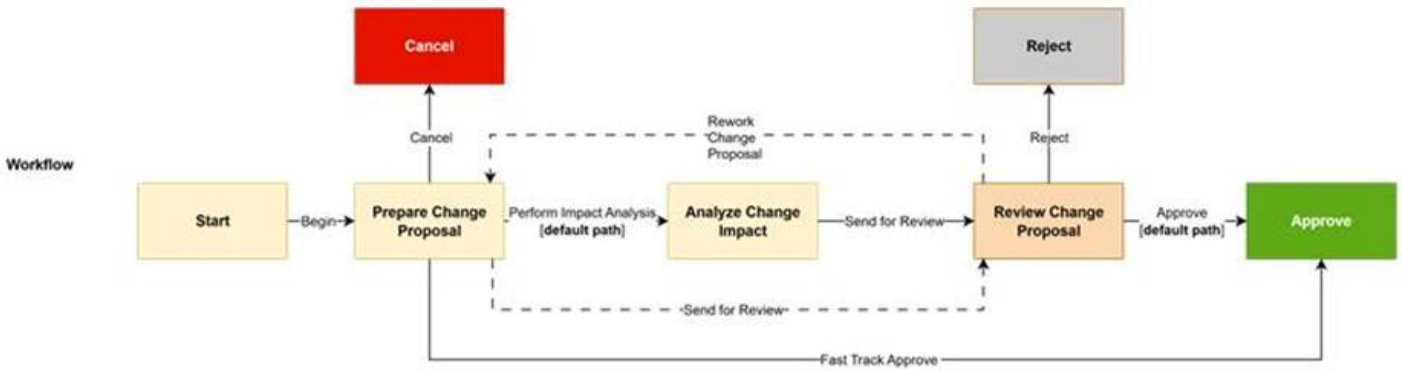
- **Draft** - The Creator submits a request with all necessary supporting details, data, and justification.
- **In Review** - The Change Review Board evaluates the proposal.
- **Approved** - The Change Request is accepted. A Change Order can be created from an Approved Change Request to implement the change.
- **Rejected** - The Change Request is declined and closed.
- **Canceled** - The Change Request is withdrawn and is no longer active. If necessary, it can be returned to the Draft state by Change Managers.
- **Closed** - The Change Request process is considered complete upon manual promotion by the Change Manager.



Change Request Workflow Overview

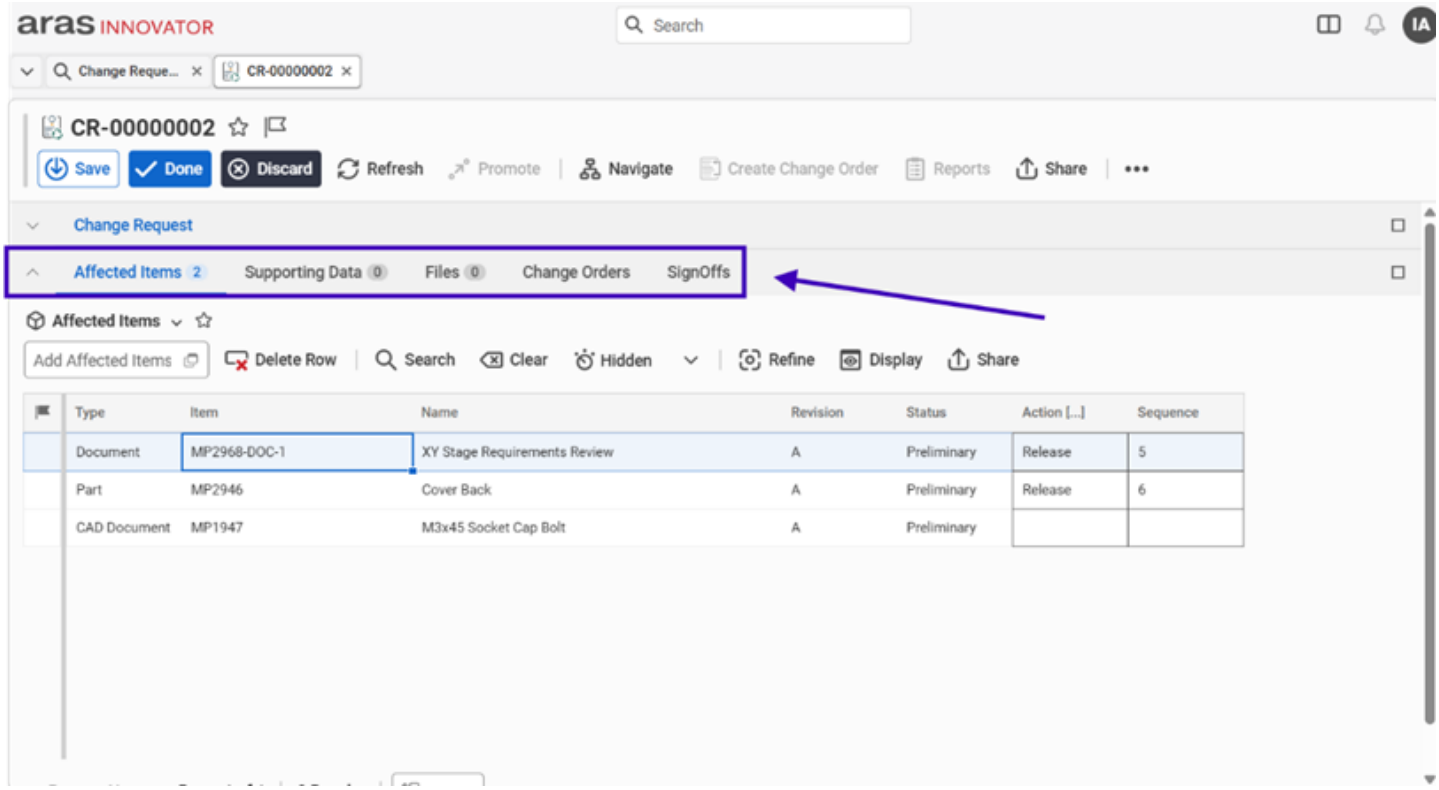
Change Requests progress through a series of activities defined within the workflow:

1. **Create Change Request:** The Creator navigates to **Contents > Unified Change Management > Change Requests > Create New Change Request**. The request is saved as a **Draft**.
2. **Prepare Change Proposal:** The Creator enters the Change Request details (e.g., Title, Reason for Change, Description, Priority, etc.). If needed, they list proposed **Affected Items** and add **Supporting Data**. Once ready, the Creator casts their vote:
    - **Perform Impact Analysis:** Moves the Change Request to the **Analyze Change Impact** step.
    - **Send for Review:** Moves the Change Request to the **Review Change Proposal** step.
    - **Fast Track Approval:** Approves the Change Request immediately if the change is minor and does not require further Impact Analysis or detailed review.
3. **Analyze Change Impact:** Change Managers or other team members review the Change Request to ensure all necessary **Affected Items** are included and assess the overall impact.
4. **Review Change Proposal:** The Change Review Board evaluates the proposal, including Impact Analysis, Affected Items, proposed actions, and planned completion date. After the review, they cast their vote:
    - **Approve**: Moves the Change Request to the **Approved** state, allowing a Change Order to be created.
    - **Rework:** Returns the plan for necessary adjustments.
    - **Reject:** Declines the Change Request.



Overview of Change Request Details

This section outlines the Change Request form and describes each tab in its view.

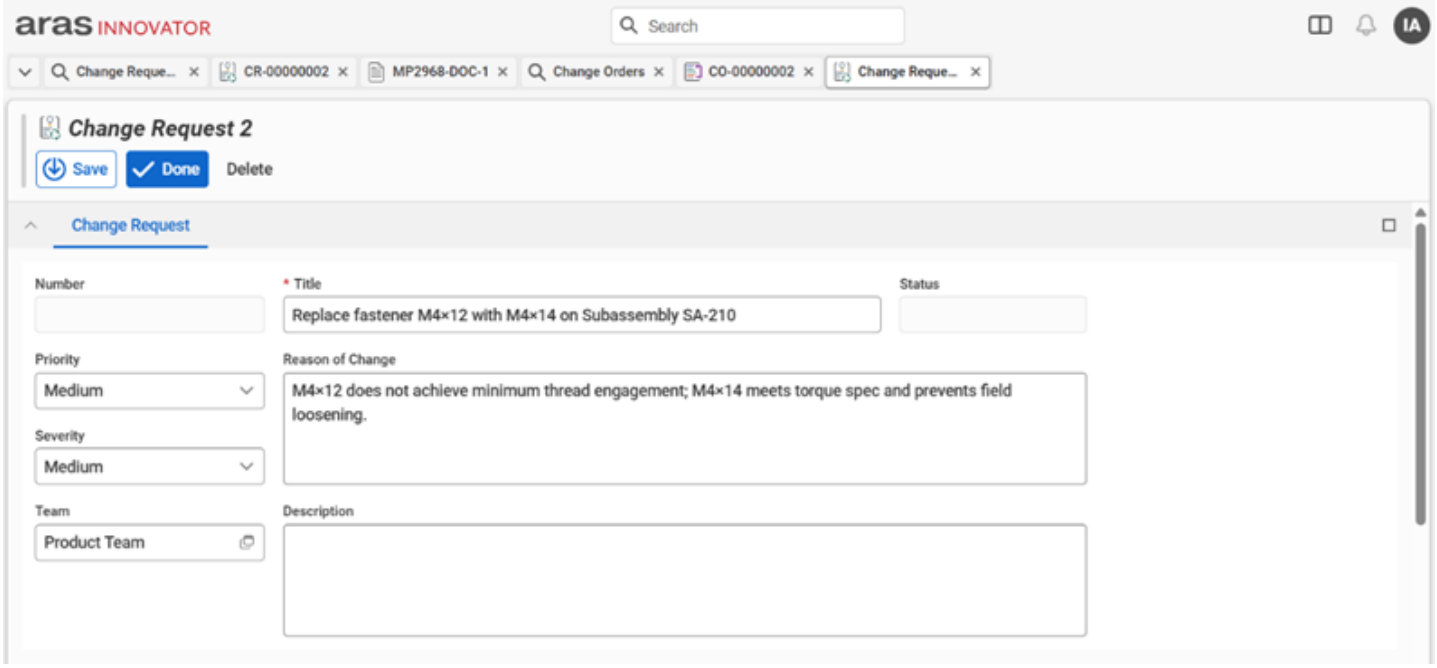


Create a Change Request

1. In the **Contents**, go to **Unified Change Management** and select **Change Requests**.
2. Click **Create New Change Request**.
3. Complete the Change Request properties, including **Title**, **Reason for Change**, and **Description**, and select the appropriate **Priority**, **Severity**, and **Team**.

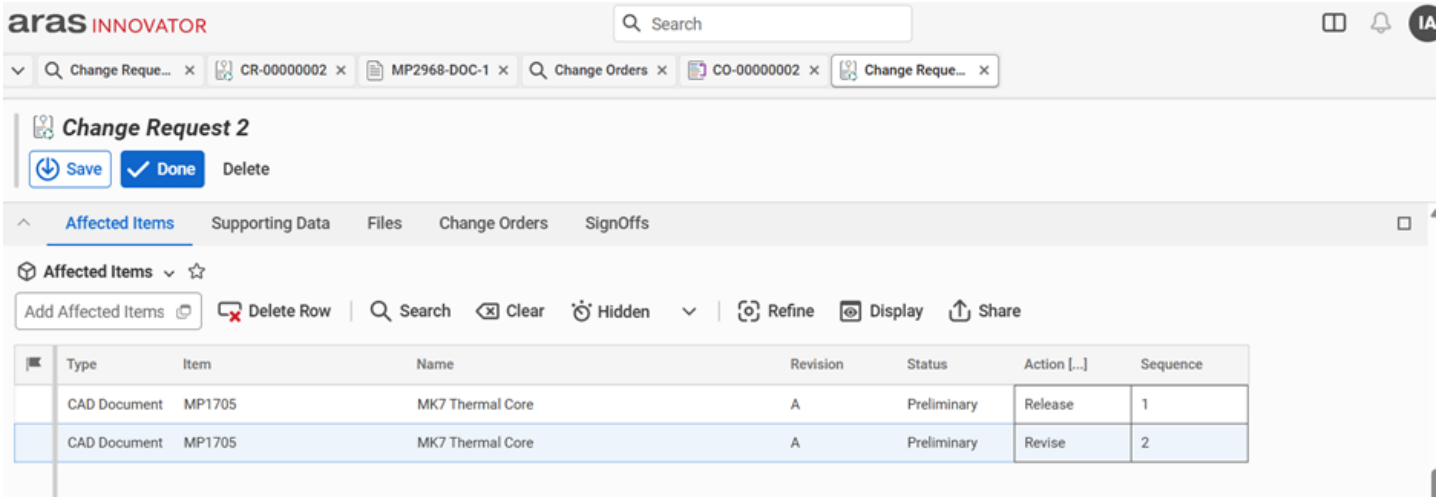
> **Important**
>
> The selected Team is used for workflow assignments.

4. Add **Affected Items**, **Supporting Data** and **Files** to scope the change.
5. Click **Save**. The Change Request will be saved with **Draft** status.



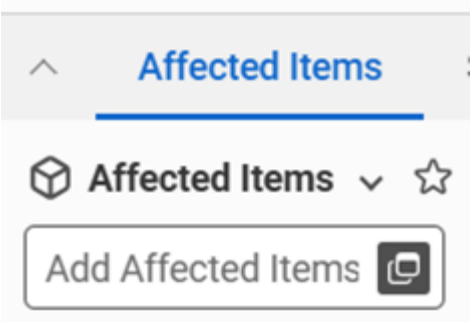
Affected Items tab

This tab displays the items for update, together with their proposed change actions (e.g., Release, Revise, Obsolete), as part of the Change Request. Items can include Parts, Documents, CAD Documents, and other change-controlled objects. The sequence of items can be arranged using the numbers in the Sequence column.



To add Affected Items to a Change Request:

1. Click **Edit** under the Change Request name.
2. In the Affected Items tab, click **the picker icon** (the small square at the right edge of Add Affected Items).



3. In the **Select Items window**, choose the item type (e.g., CAD Documents, Parts) and pick the item(s) to add to the Change Request.

4. Click **OK** to add it.

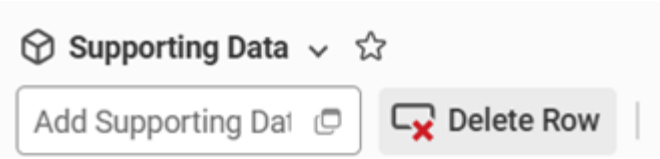
To replace or remove Affected Items in a Change Request:

1. Navigate to **the Affected Items tab** in the Change Request.
2. **Right-click** on the Affected Item to remove or replace.
3. Choose **Remove** or **Replace** from the menu.

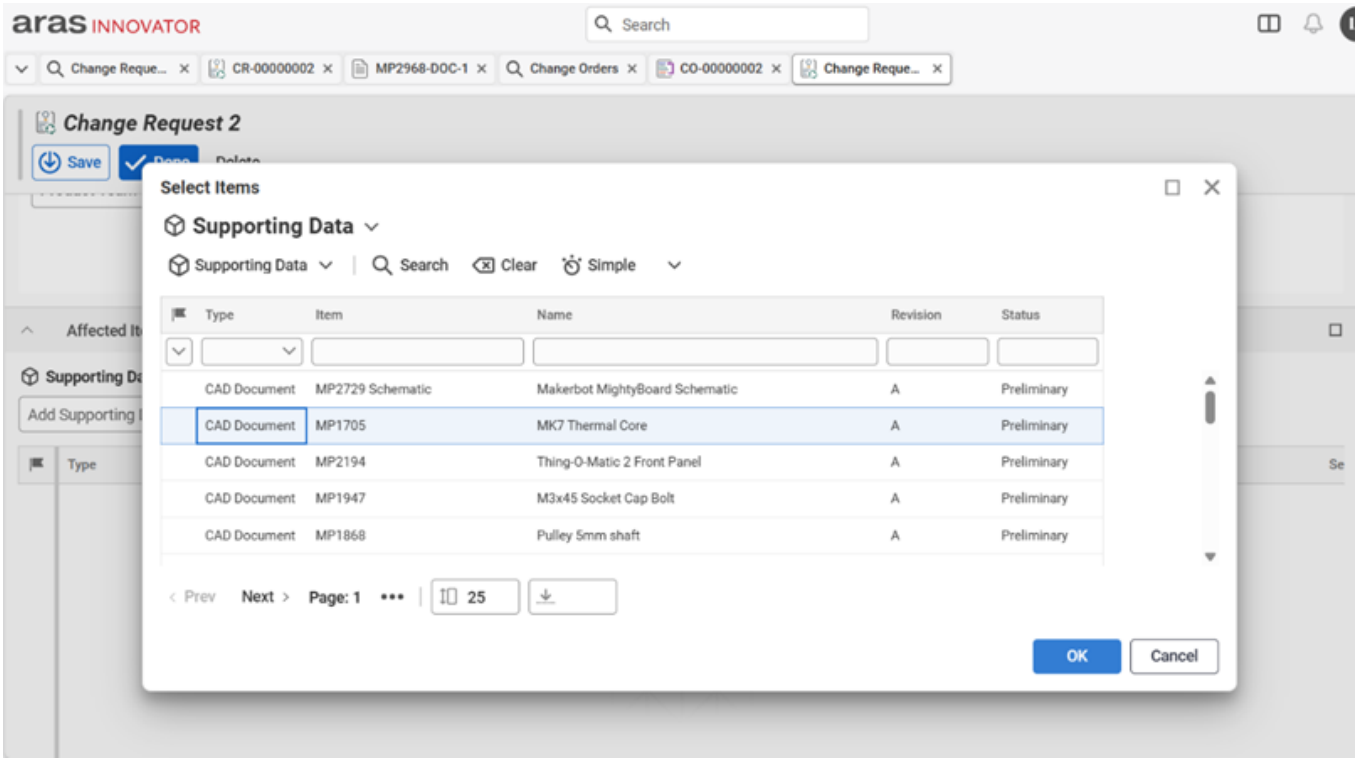
Supporting Data tab

The **Supporting Data** tab collects all evidence, documentation, and references related to a Change Request, helping stakeholders understand the rationale, assess impact, and make informed decisions.

To add Supporting Data:



1. Click the **Edit** button under the Change Request name.
2. To add an item as supporting date, click **the square picker icon** to open the selection dialog.
3. Select item(s) from the list and click **OK**.
4. Enter comments for each supporting data item, as needed.



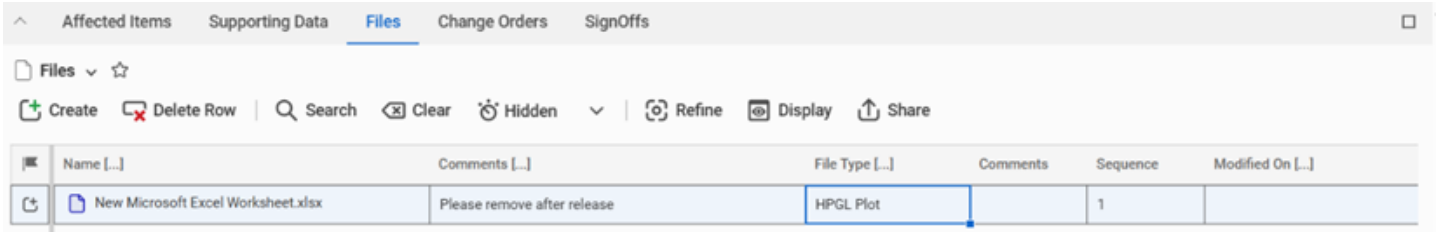
To remove Supporting Data:

5. Select the item, click Delete Row.

Files tab

File Attachments allow users to add images, spreadsheets, or other files that provide additional information or context for the Change Request.

- To add a File, click the **Create button** in the Files tab when editing the Change Request.
- To delete a File, select the row(s) to delete and click **Delete Row**.



Change Orders tab

This tab links the Change Request to its execution by connecting it to the Change Order(s) that will implement the approved changes. A simple Change Request may result in a single Change Order, while a broader or more complex request can be divided into multiple Change Orders.

The list of Change Orders linked to a Change Request is view-only. To associate a Change Request (CR) with a Change Order (CO), one of the following actions can be performed:

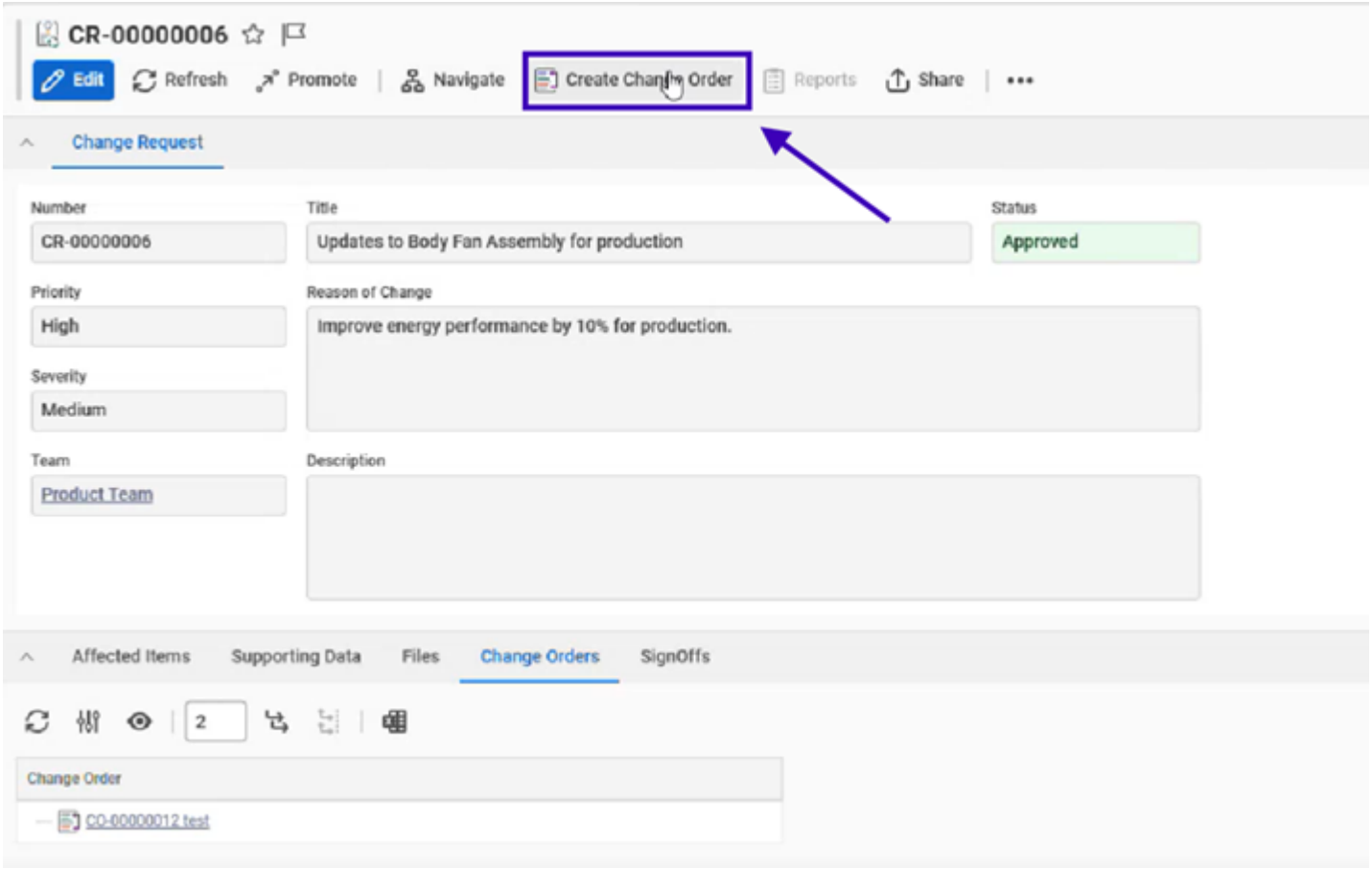
- Use the **Create Change Order** button from an approved CR. This creates the CO and establishes the relationship with the CR.
- Open an existing CO, navigate to the **Change Requests** tab, then search for and add the CR to the CO.

SignOffs tab

SignOffs capture the signoff matrix, including activities, assignees, status, completion details, timestamps, voting results, and comments.

Create a change order from an approved change request

When a Change Request is approved, the **Create Change Order** button becomes available. Selecting it carries forward the affected items and change actions into the Change Order create form automatically.



1. Click **Create Change Order** to open a new Change Order form, prefilled with the Change Request's affected items and actions.
2. Review and edit the Change Order details as needed.
3. Click **Save**. All standard validations configured for new Change Orders—including those on the affected items—will be applied.

Delete a Change Request

A Change Request can only be deleted by the Creator when the item is in the Draft state of the lifecycle.

There are two ways to delete a Change Request:

1. Go to the list of Change Requests **(Contents → UCM → Change Requests)**, and right-click on the Change Request you want to delete. Click **More** and then click **Delete**.
2. Click the three-dot icon in the Change Request view and select **Delete**.

Change Orders

Overview

A **Change Order** is a process used to communicate and track the approval and implementation of changes to a product or process. It serves as a follow-up to a Change Request, ensuring that approved changes are executed effectively and consistently.

If Change Requests (CRs) are the front door of change management in PLM, then Change Orders (COs) are the execution mechanism.

Purpose of Change Orders

A Change Order (CO) is used to formalize, approve, and execute product changes after a Change Request has been reviewed and accepted. It ensures that the approved change is implemented in a controlled, traceable, and coordinated manner across all affected areas. The following are common use cases for initiating a Change Order:

- **Implementing approved changes** - After a Change Request has been evaluated and approved, a Change Order specifies what exactly will change and how the change will be executed.
- **Coordinating cross-functional execution** - Ensures engineering, manufacturing, quality, supply chain, and service teams all act on the same approved change instructions.
- **Managing revisions** - Defines updated revisions of parts, documents, software, or processes that need to be released.

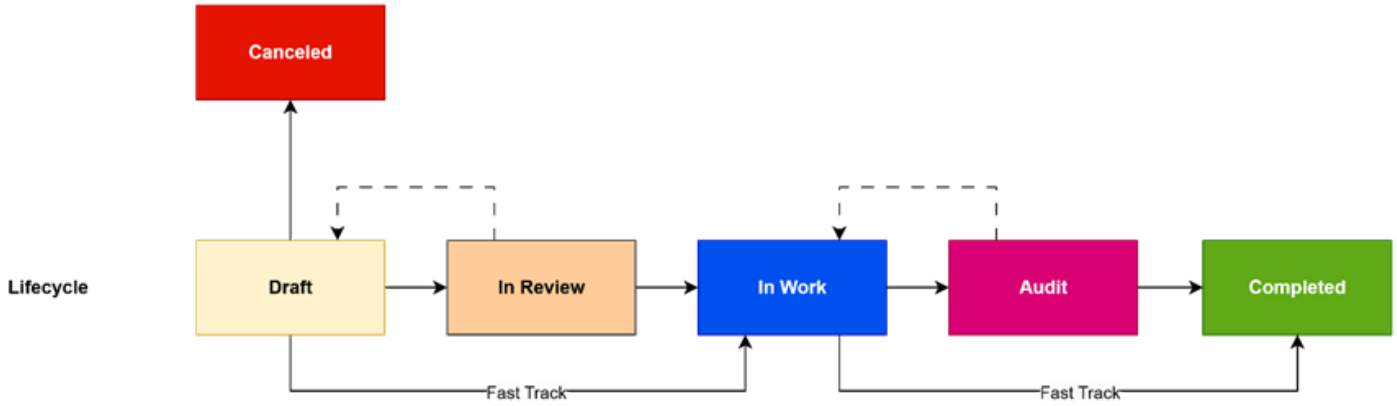
Key Roles in the Change Order Process

- **Change Authors** - Creates new change orders.
- **Change Viewers** - Tracks the status and progress of changes without the ability to edit.
- **Change Manager** - Coordinates changes and manages escalations.
- **Change Administrators** - Configures and oversees the change management process, assists users in case of errors, and resolve exception situations.
- **Team Member** - Executes the change plan and updates the affected items.
- **Change Review Board** - Reviews the implementation plan.
- **Change Verifiers** - Ensure implemented changes have been correctly applied and meet defined requirements.

Change Order Lifecyle Overview

Change Orders move through different stages, which are updated automatically by the system as the workflow progresses:

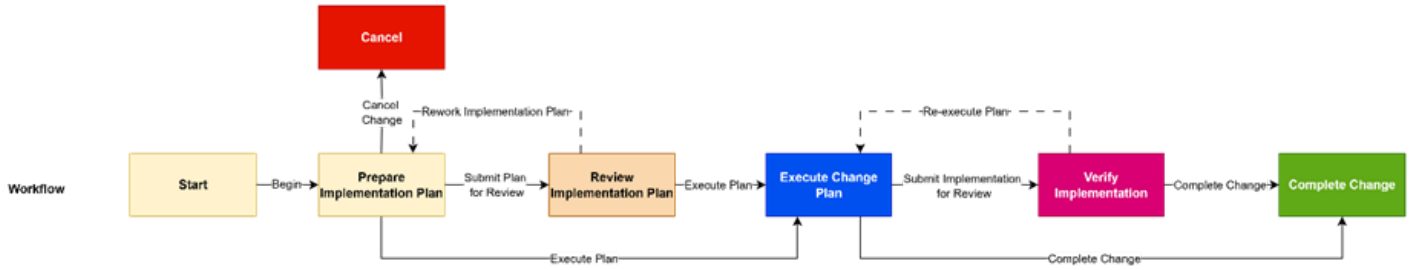
- **Draft** - The Change Order is first created and defined.
- **In Review** - The plan has been submitted for review.
- **In Work** - The plan is being carried out.
- **Audit** - An audit is required before completion.
- **Completed** - The Change Order is finished and closed.
- **Canceled** - The Change Order is no longer needed and is stopped.



Change Order Workflow Overview

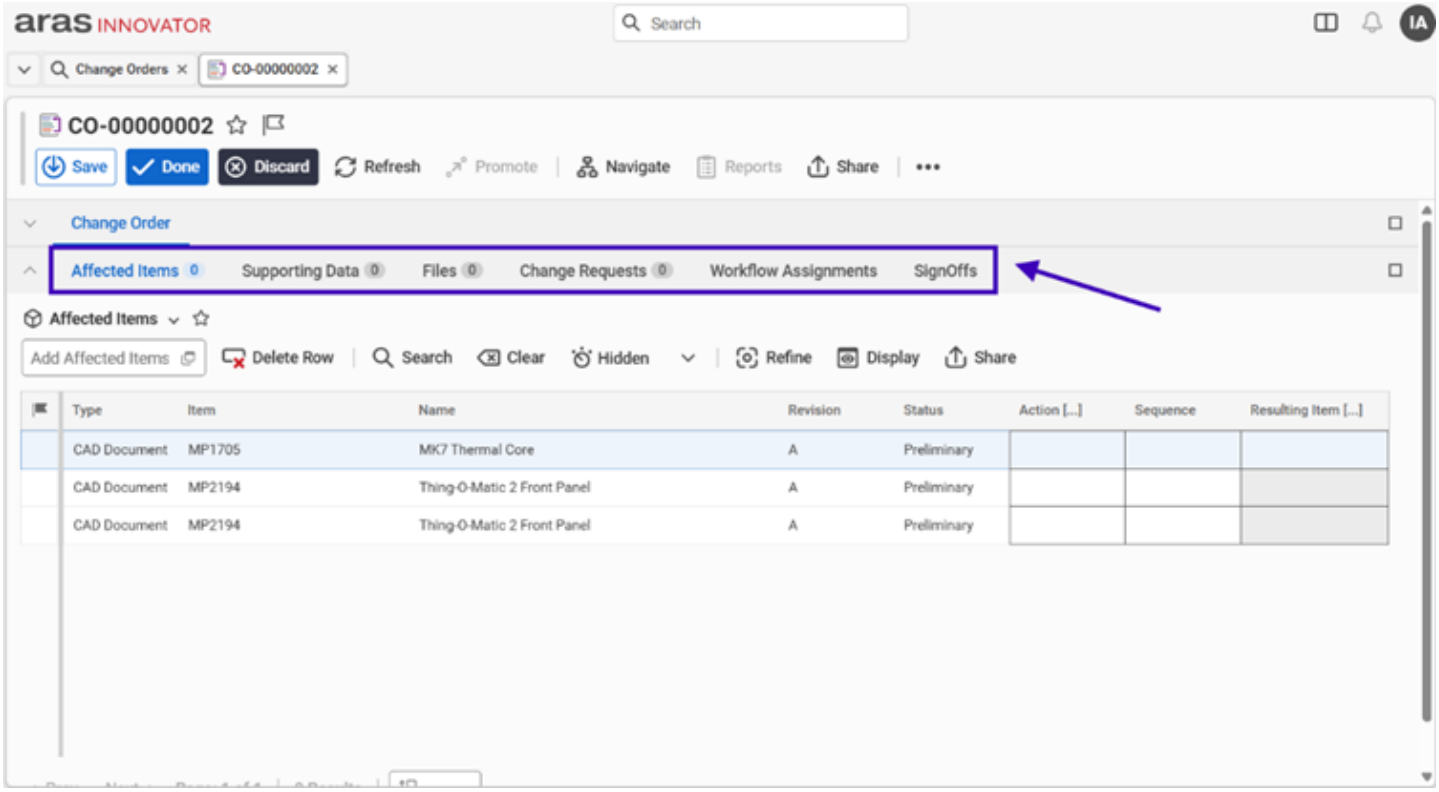
Change Orders progress through a series of activities defined within the workflow:

1. **Create Change Order**: The Creator navigates to **Contents > Unified Change Management > Change Orders > Create New Change Order**. The order is saved as a Draft.
2. **Prepare Implementation Plan**: The Creator enters Change Order details (e.g., Title, Team, Change Manager, Planned Completion Date). They list all Affected Items and choose the required change actions. A manual impact check can be performed to cover dependencies (e.g., add a parent assembly when retiring a part so its BOM can be updated). Once ready, the Creator casts their vote:
    - **Submit Plan for Review**: Moves the Change Order to the Review Implementation Plan step.
    - **Execute Plan**: Skips review and moves directly to Execute Change Plan.
    - **Cancel Change**: Sets the Change Order to Canceled if it's no longer required.
3. **Review Implementation Plan**: The Change Review Board evaluates the plan, including Affected Items, selected actions, dependencies, and timeline. They cast their vote:
    - **Approve**: Moves the Change Order to Execute Change Plan.
    - **Rework**: Returns the plan for adjustments.
    - **Reject**: Terminates the Change Order.
4. **Execute Change Plan**: Assigned Team Members implement the change: they update Affected Items (content, properties, and relationships) and carry out any Change Tasks. When finished, they cast their vote:
    - **Submit for Audit**: Sends the executed plan to Verify Implementation.
    - **Complete Change**: Moves to change completion without additional review.
5. **Verify Implementation**: Change Verifiers review the results of execution for technical accuracy and completeness. They cast their vote:
    - **Complete Change**: Approves the implementation and finalizes the Change Order.
    - **Re-execute Plan**: Returns the change order to Execute Change Plan for corrections.
6. **Complete Change**: The Change Order is finalized; all tasks are closed and the record is set to Completed.



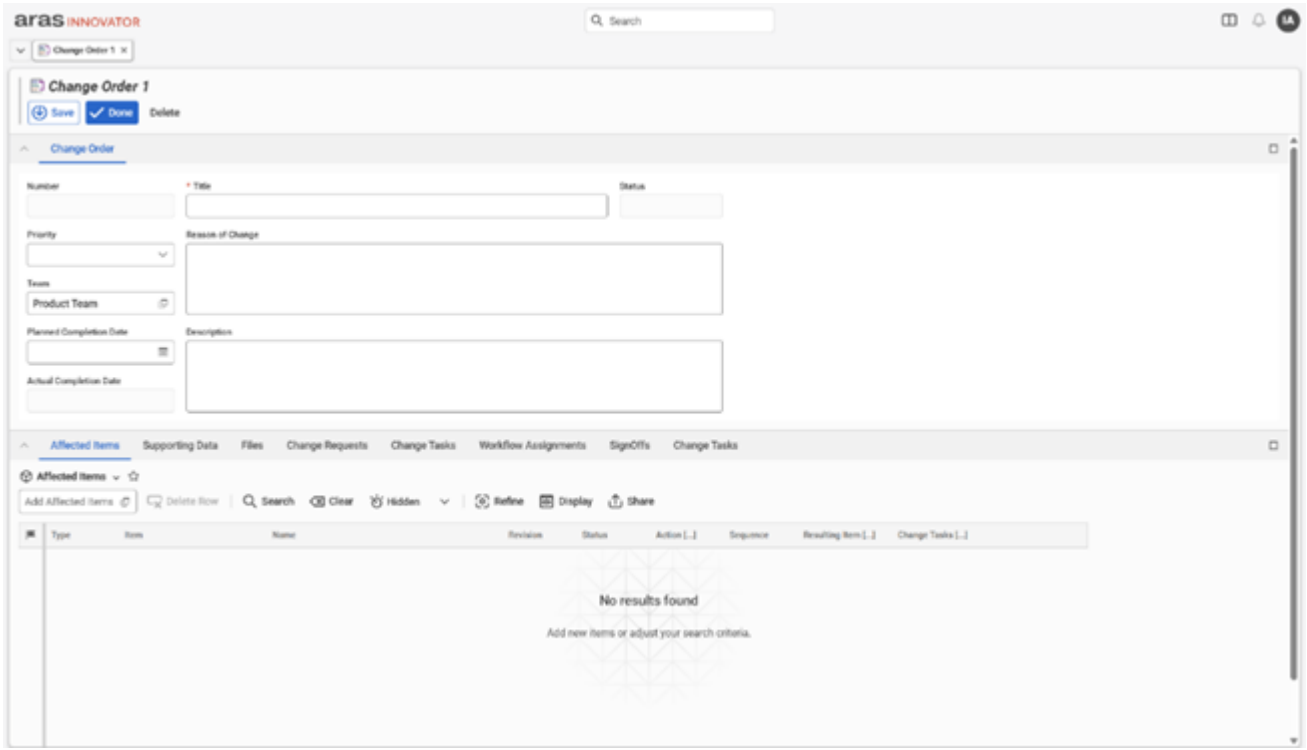
Overview of Change Order Details

Below is an overview with an explanation of each tab inside the Change Order view.



Create a Change Order

1. In the **Contents**, go to **Unified Change Management** and select **Change Orders**.
2. Click **Create New Change Order**.
3. Complete the Change Order properties, such as **Title**, **Reason of Change**, **Description**, **Priority** and the **Completion Date**.
4. Click **Save**.

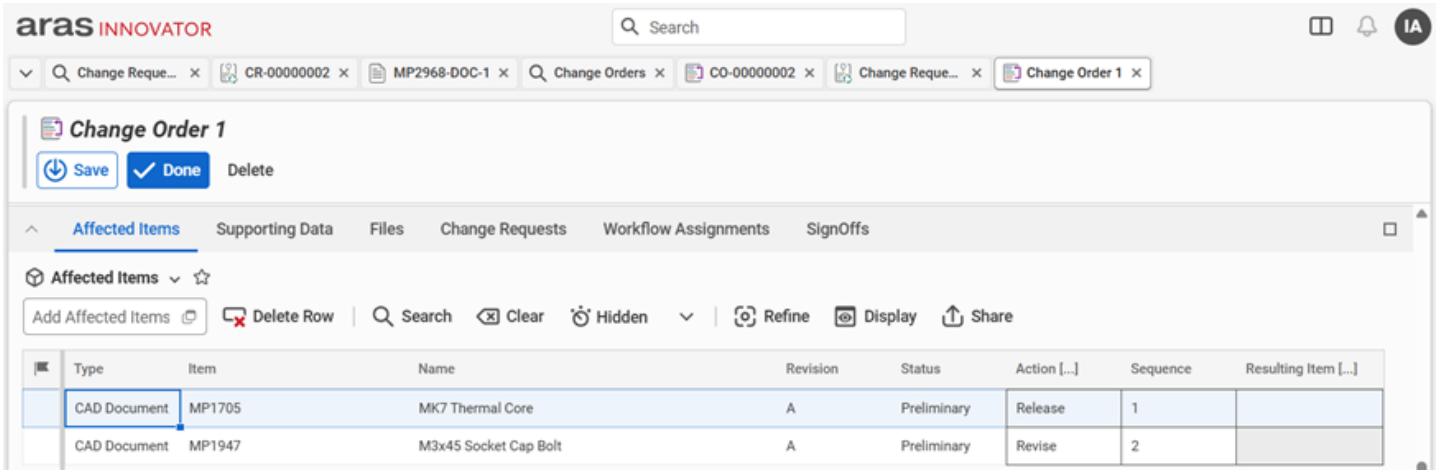


Affected Items tab

This tab lists the Parts, CAD Documents, and other change-controlled items that the Change Order will execute. Each row displays the intended **Action** (Release, Revise, Obsolete). The sequence of items can be managed using the numbers in the **Sequence** column. **Revision** and **Status** display information from the affected item.

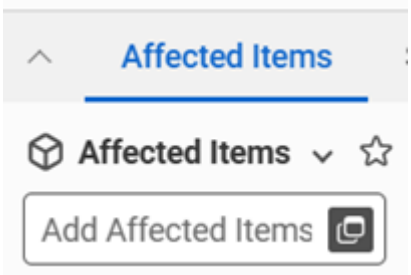
The behavior of pre-configured actions are as follows:

- **Release**: Publishes a preliminary item for the first time via a Change Order. When the Change Order is done, the item moves from *Preliminary* to *Released*.
- **Revise**: Creates a new version of an item that's already released. The new version becomes *Released*, and the old one changes to *Superseded*.
- **Obsolete**: Marks the item as *Obsolete* when the Change Order is completed.
  The **Resulting Item** column displays an item link according to the following logic:
- While the Change Order is open:
  - If the change action has **not yet been executed**, it links to the latest version of the affected item.
  - If the change action **has been executed**, it links to the latest revision/version of the item.
- When the Change Order is completed:
  - It links to the item revision/version as of the time of Change Order closure.



Add Affected Items to a Change Order

1. Click **Edit** under the Change Order name.
2. In the Affected Items tab, click **the picker icon** (the small square at the right edge of Add Affected Items).



3. In the **Select Items window**, choose the item type (e.g., CAD Documents, Parts) and pick the item you need.
4. Click **OK** to add it.
5. Select the **Action** either by typing the action name or by choosing it from the search results.
6. Enter the item **Sequence** to set its order; otherwise, the system assigns the next available sequence.

Replace or Delete an Affected Item from a Change Orders

1. Go to **the Affected Items tab** in the Change Order.
2. **Right-click** on the Affected Item to remove or replace.
3. Choose **Remove** or **Replace** from the menu.

Use Impact Analysis Tool

1. Go to **the Affected Items tab** in the Change Order.
2. **Right-click** on the Affected Item to open Impact Analysis Tool.
3. Choose **Impact Analysis Tool** from the menu.

> **Important**
>
> Impact Analysis Tool is enabled when Change Item is in Edit mode only.

4. Different Impact Analysis Views can be selected using the dropdown in the toolbar.
5. **Right-click** on any row to open Add to Change.
6. Choose **Add to Change** from the menu. The selected Item will be added as the Affected Item back to the Change Item.

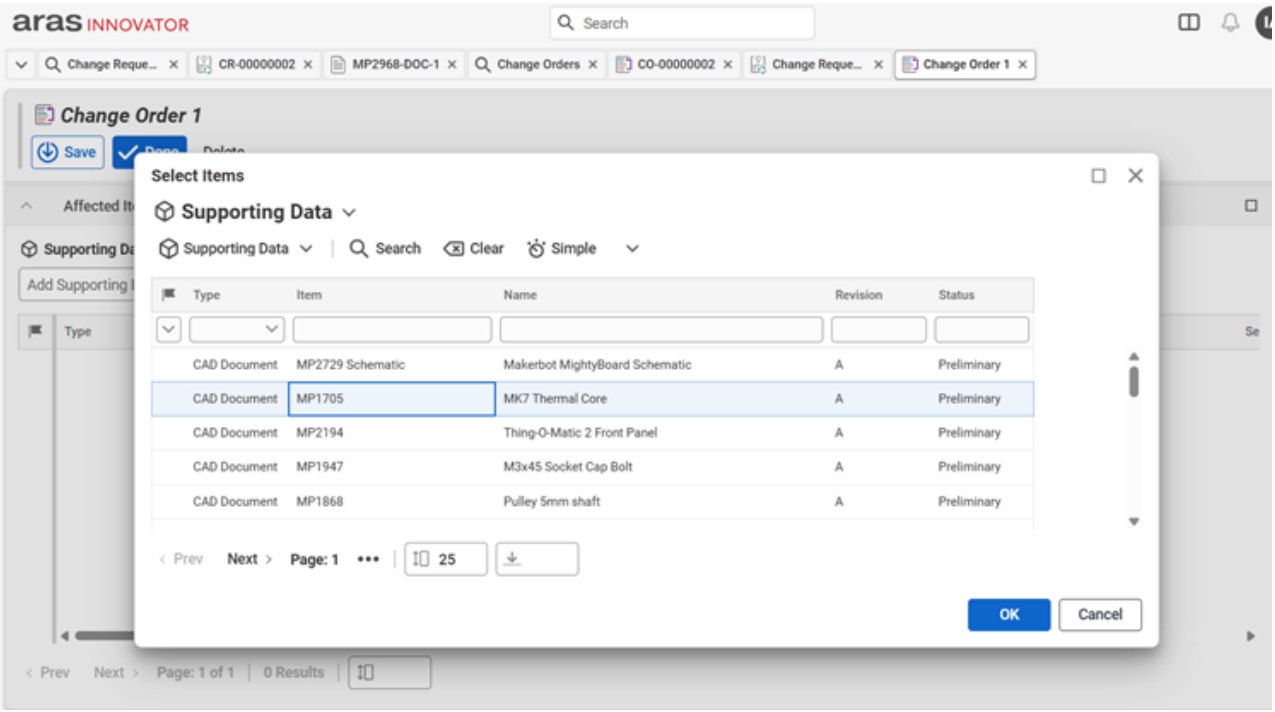
Supporting Data tab

Supporting Data contains the evidence and implementation details approvers need. It complements what was proposed on the originating Change Request, so the execution record is self-contained.

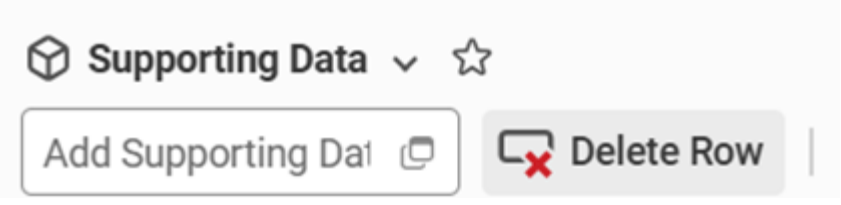
Add or remove Supporting Data:

1. Click the **Edit** button under the Change Order name.
2. Click **the square picker icon** to open the selection dialog.

   - To **add Supporting Data**, select the document from the list and click **OK**.



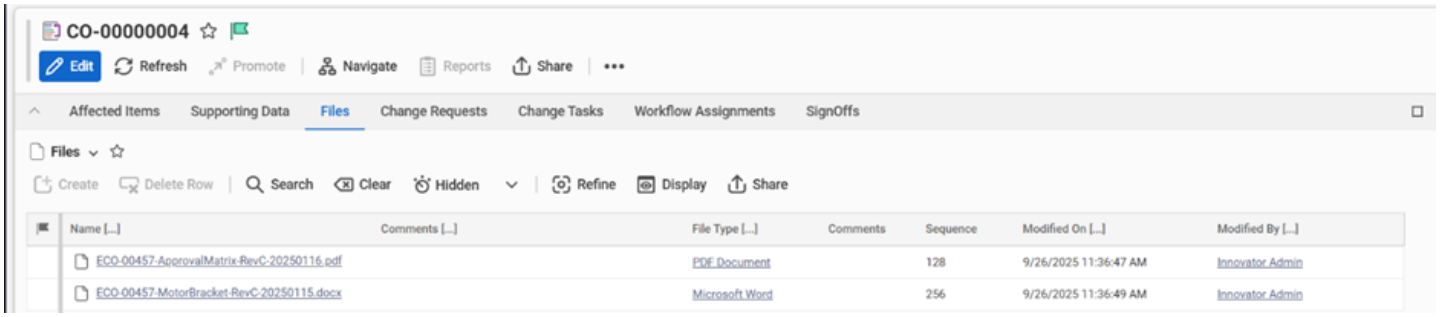
   - To **delete Supporting Data**, click **Delete Row**.



Files tab

The Files tab holds quick attachments tied to the Change Order, such as screenshots, redlines, PDFs, or temporary spreadsheets that explain the change.

- To add a File, click the **Create button** in the Files tab when editing the Change Order.
- To delete a File, click **Delete Row** when you select the File you want to remove.



Change Requests tab

This tab links the Change Order back to the authorizing Change Request(s). Simple cases may reference a single Change Request, while broader implementations can point to multiple Change Requests consolidated into one Change Order.

To associate a Change Request (CR) with a Change Order (CO), one of the following actions can be performed:

- From this tab, search for and add an existing CR to the CO.
- Use the **Create Change Order** button from an approved CR. This creates the CO and establishes the relationship with the CR.

SignOffs tab

SignOffs capture the signoff matrix, including activities, assignees, status, completion details, timestamps, voting results, and comments.

Delete a Change Order

A Change Order can only be deleted by the Creator when the item is in the Draft state of the lifecycle. There are two ways to delete a Change Order:

1. Go to the list of Change Requests (**Contents** → **UCM** → **Change Orders**), and right-click on the Change Order you want to delete. Click **More** and then click **Delete**.
2. Click the three-dot icon in the Change Order view and select **Delete**.

Change Tasks

Overview

Change Tasks are the action units of a Change Order. They ensure that all required updates across engineering, manufacturing, supply chain, quality, compliance, and service are explicit, assigned, tracked, and completed before the change is implemented.

Important

Change Tasks can only be created within Change Orders, not as standalone items in the system.

Purpose of Change Tasks

Change Tasks add value by breaking significant changes into manageable actions, assigning clear ownership to individuals or teams, and providing status visibility within the Change Order. They reduce execution risk by preventing missed steps and creating full traceability of who did what, and when.

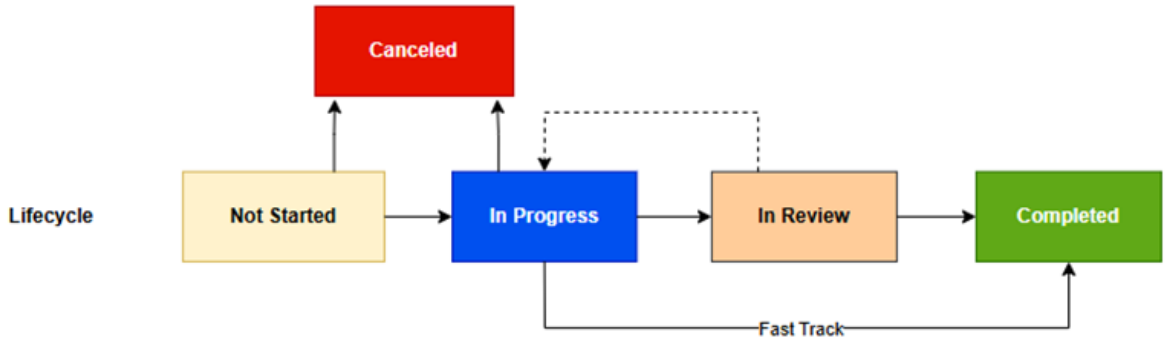
Key Roles in the Change Task Process

- **Change Authors** - Creates new Change Tasks.
- **Change Viewers** - Tracks the status and progress of changes without the ability to edit.
- **Team Member** - Implements the approved changes to the affected items.
- **Team Manager** - Reviews completed implementation.

Change Task Lifecycle Overview

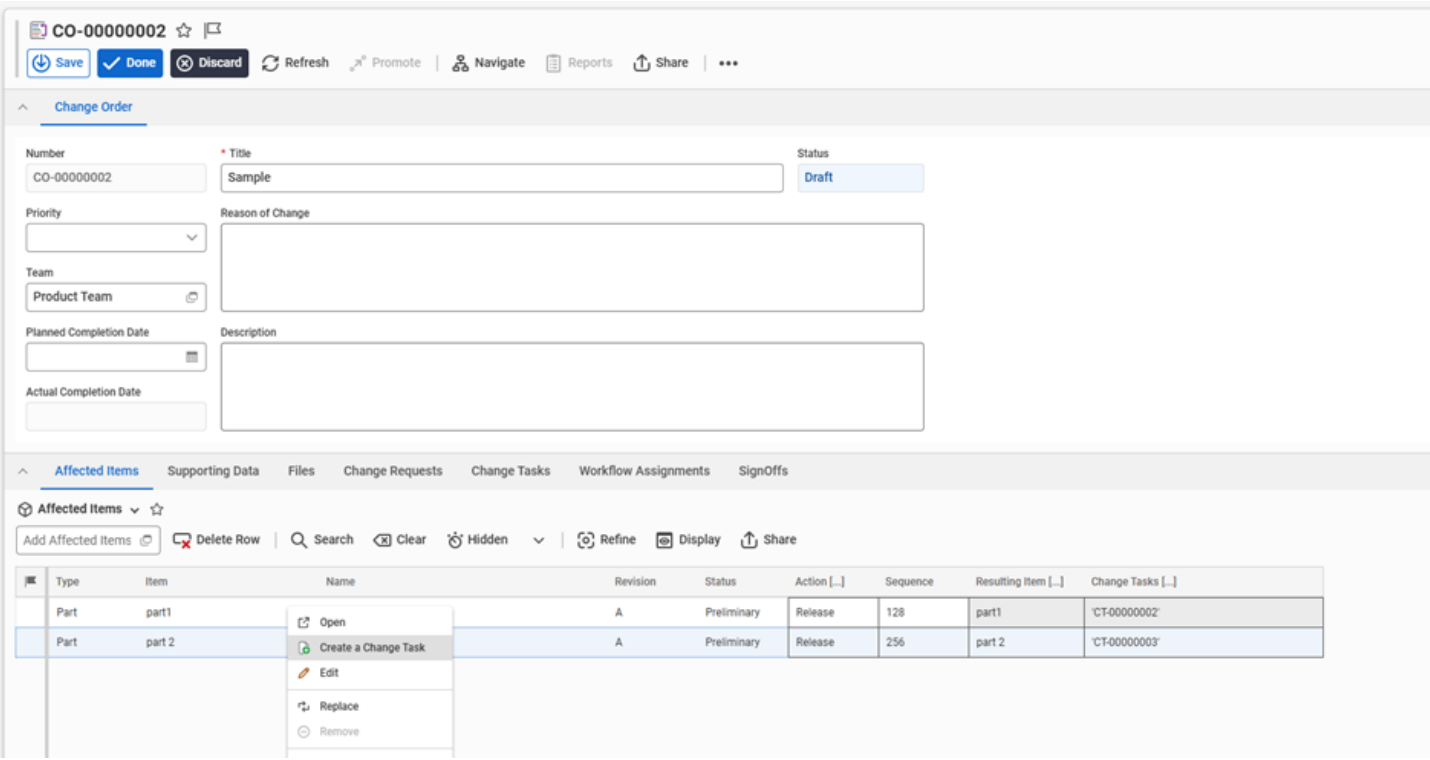
Change Tasks move through different stages, which are updated automatically by the system as the workflow progresses:

- **Not Started** - The Change Task is first created and defined.
- **In Progress** - The change is in progress; the assignee works with the Affected Items.
- **In Review** - The change required to be reviewed.
- **Completed** - The Change Task is finished and closed.
- **Canceled** - The Change Task is no longer needed and has been stopped.

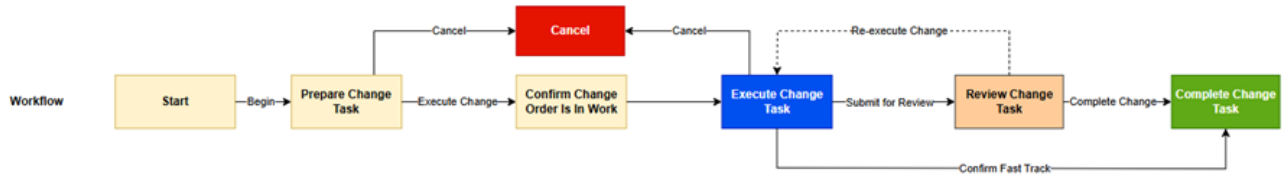


Change Task Workflow Overview

Below is an overview of creating a Change Task and managing the workflow:

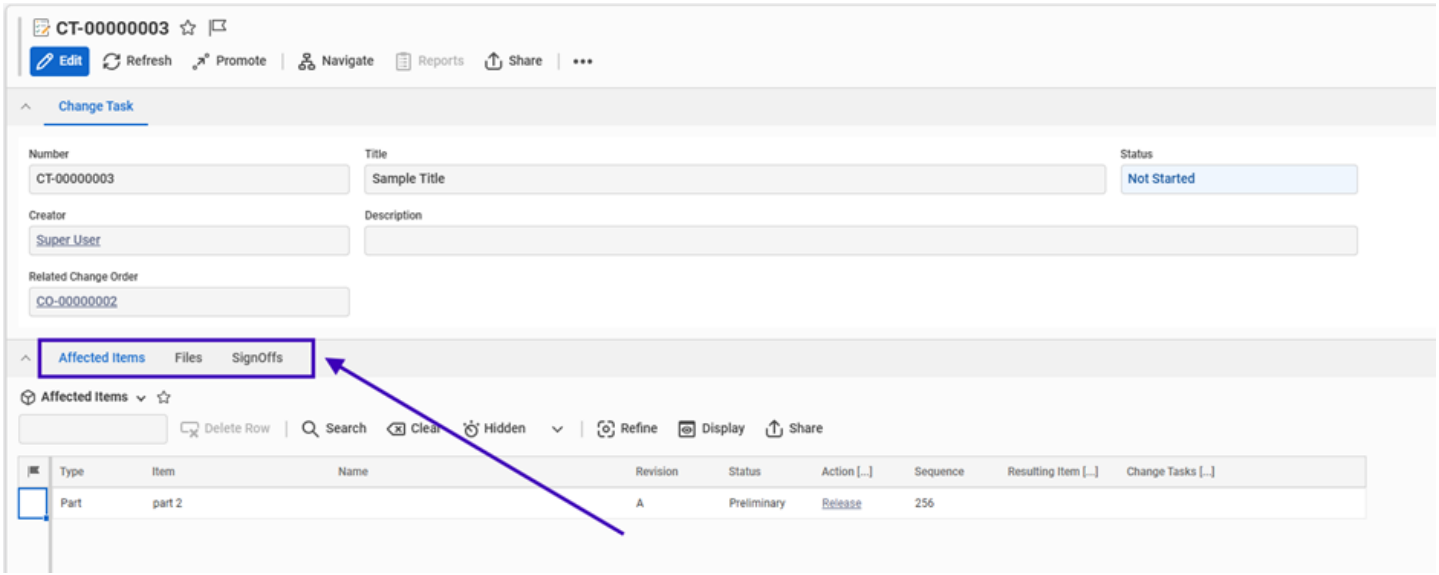


1. The Creator navigates to **Contents** → **Unified Change Management** → **Change Orders** → click **Edit** → go to the **Affected Items** tab and right-click on one or more → click **Create a Change Task**.
2. **Prepare Change Task**: The Creator enters the Change Task Title. The **Affected Items** tab is prepopulated with an Affected Item(s) selected in Change Order. The Creator can also add more Affected Items from the Change Order if needed. Once ready, the Creator casts their vote:
    - **Execute Change**: Moves the Change Task to the **Confirm Change Order Is In Work** step.
    - **Cancel**: Sets the Change Task to **Canceled** if it's no longer required.



3. **Confirm Change Order Is In Work**: It is a stage where Change Tasks wait for Change Orders to be in the **In Work** state. Once the Change Order goes to In Work, the Change Task automatically goes to the next stage. Manual votes are not needed.
4. **Execute Change Task**: Assigned Team Members implement the change. They update Affected Items (content, properties, and relationships). Once finished, they cast their vote:
    - **Submit for Review**: Sends the executed change to Verify Implementation.
    - **Confirm Fast Track**: Moves directly to completion without additional review.
    - **Cancel**: Sets the Change Task to Canceled if it's no longer required.
5. **Review Change Task**: Team Managers review the implementation of the changes. They cast their vote:
    - **Complete Change**: Approves the implementation and finalizes the Change Task.
    - **Re-execute Change**: Sends the task back to Execute Change Task for corrections.
6. **Complete Change**: The Change Task is finalized.

Overview of Change Task Details



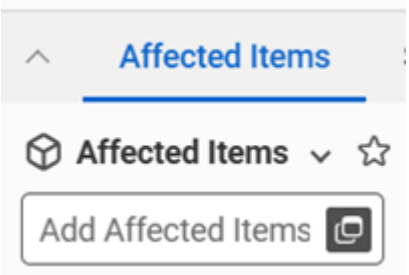
Affected Items tab

This tab lists the Parts, CAD Documents, and other change-controlled Items that could be changed in this Change Task. Each row shows the intended Action (Release, Revise, Obsolete) and its sequence number, which determines the order in which the items appear in the list. Revision and Status reflect the current state. The **Resulting Item** shows the latest version of an Affected Item while the Change Task is active. When the Change Task moves to **Cancelled** or **Completed**, the Resulting Item stops updating and instead shows the most recent version of the Affected Item at the moment the task became inactive.

All columns are already filled in with information from the related Change Order. You can choose an Affected Item from that Change Order and adjust the **Sequence** field if you need to.

Add Affected Items to Change Task

1. Click **Edit** under the **Change Task** name.
2. In the Affected Items tab, click **the picker icon** (the small square at the right edge of Add Affected Items).



3. In the **Select Items window**. The presented list shows the affected items on the Change Orders. Choose the item type (e.g., CAD Documents, Parts) and pick the item you need.
4. Click **OK** to add it.

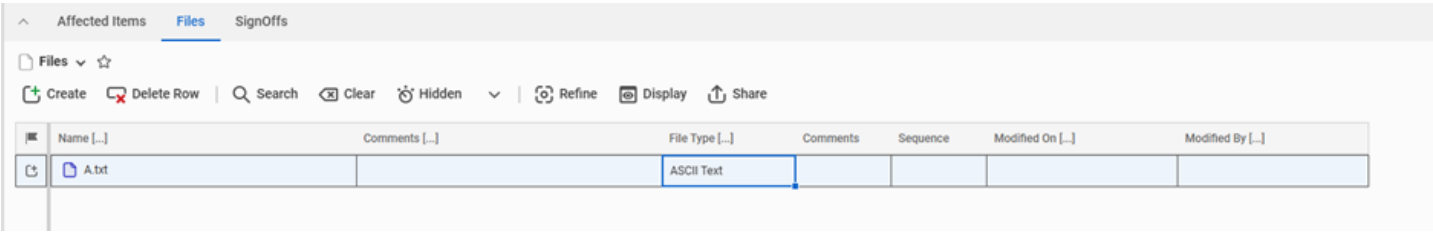
Replacing and deleting an Affected Item in Change Tasks

1. Go to the **Affected Items tab** in the Change Task.
2. **Right-click** on the Affected Item you want to remove or replace.
3. Choose **Remove** or **Replace** from the menu.

Files tab

The Files tab holds attachments tied to the Change Task, such as screenshots, red lines, PDFs, or temporary spreadsheets that explain the change.

- To add a File, click the **Create button** in the Files tab when editing the Change Task.
- To delete a File, click **Delete Row** when you select the File you want to remove.



SignOffs tab

SignOffs capture the signoff matrix, including activities, assignees, status, completion details, timestamps, voting results, and comments.

Working with Legacy Change Management

Overview

In environments where Unified Change Management (UCM) and legacy Product Engineering (PE) Change Management coexist, a few compatibility rules help keep items consistent and avoid conflicts.

Key compatibility rules

- Items managed by Change Orders cannot be manually promoted or revised until the Change Order is completed or canceled.
- Items in active PE Changes cannot be added to Change Orders. If you attempt this and see a validation error, you must complete or cancel the PE Change Item or remove the item from the Affected Items.
- Legacy PE Changes can update items previously managed by UCM. Authorized PE Change Items can promote or revise items from completed UCM Change Orders.

Changes pending flag behavior

Known behavior: changes pending flag

The has_change_pending flag is a legacy indicator from PE Change Management that shows when an item is part of an active change process.

In Unified Change Management

- The flag remains for compatibility but is not used by UCM.
- When a Change Order is promoted to In Work, the flag is set on all affected items but may be cleared during versioning.
- This behavior is expected and does not affect UCM.
- Use the Changes tab of the Part, CAD, or Document to track active changes instead of relying on the flag.

Important

If you need to check whether an item is part of an active change, open the **Changes** tab. It shows all UCM and PE Change Items that currently affect the item (image below).