# Aras Digital Twin Core 12.0R3

## Administrator Guide

***Document #: 12.0R3.2021030203***

***Last Modified:*** *12/22/2021*

# Copyright Information



Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810

**Phone:** 978-806-9400
**Fax:** 978-794-9826

**E-mail:** support@aras.com
**Website:** https://www.aras.com

# Table of Contents

# Send Us Your Comments

Aras Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for future revisions.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where and what level of detail?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, indicate the document title, and the chapter, section, and page number (if available).

You can send comments to us in the following ways:

**Email:**
TechDocs@aras.com
Subject: Aras Product Documentation

Or,

**Postal service:**
Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Attention: Aras Technical Documentation

If you would like a reply, provide your name, email address, address, and telephone number.

If you have usage issues with the software, visit https://www.aras.com/support/

# Document Conventions

The following table highlights the document conventions used in the document:

**Table 1:** Document Conventions

| Convention | Description |
|---|---|
| **Bold** | This shows the names of menu items, dialog boxes, dialog box elements, and commands.<br>Example: Click **OK**. |
| `Code` | Code examples appear in `courier` font. It may represent text you type or data you read. |
| `Yellow highlight` | Code highlighted in yellow draws attention to the code that is being indicated in the content. |
| `Yellow highlight with red text` | Red text highlighted in yellow indicates the code parameter that needs to be changed or replaced. |
| *Italics* | Reference to other documents. |
| **Note:** | Notes contain additional useful information. |
| **Warning** | Warnings contain important information. Pay special attention to information highlighted this way. |
| Successive menu choices | Successive menu choices may appear with a greater than sign (-->) between the items that you will select consecutively.<br>Example: Navigate to **File** --> **Save** --> **OK**. |

# 1 Overview

The Aras Digital Twin Core (DTC) application is configured to work out of the box. This document describes the options and procedures for customizing the application as well as its configuration requirements.

For detailed information on the DTC application concepts and features, refer to the *Aras Digital Twin Core 12.0R3 – User Guide*.

All the application functionality is available via the Aras Innovator 12 User Interface (UI). Dedicated subsections cover particulars of using AML with the application. You should always meet the requirements of these specifics when working with the application via AML.

The **php_** prefix denotes items such as Methods, Permissions, UI elements, and so on, that belong to the DTC application data model and architecture.

# 2 Configuring Asset Identities

This section discusses the Identities that manage the Digital Twin Core (DTC) application, the application requirements, and their out-of-the-box configuration.

DTC 12.0R3 has built-in Identities that can be grouped as follows:

- Asset Identities
- Operational Event Identities

## 2.1 Asset Identities

To hold specific Access Rights within the Asset Permissions for the Asset Items, the DTC application has the following built-in Group Identities that are referred to as the Asset Identities:

- **Asset Viewer**—an observer of all DTC Items with view-only abilities.
- **Asset User**—a regular user of the Asset Items with limited editing abilities.
- **Asset Editor**—an advanced user of the Asset Items with limited creation, editing, and deletion abilities.
- **Asset Admin**—an administrator of the Asset Items with full creation, editing, and deletion abilities.

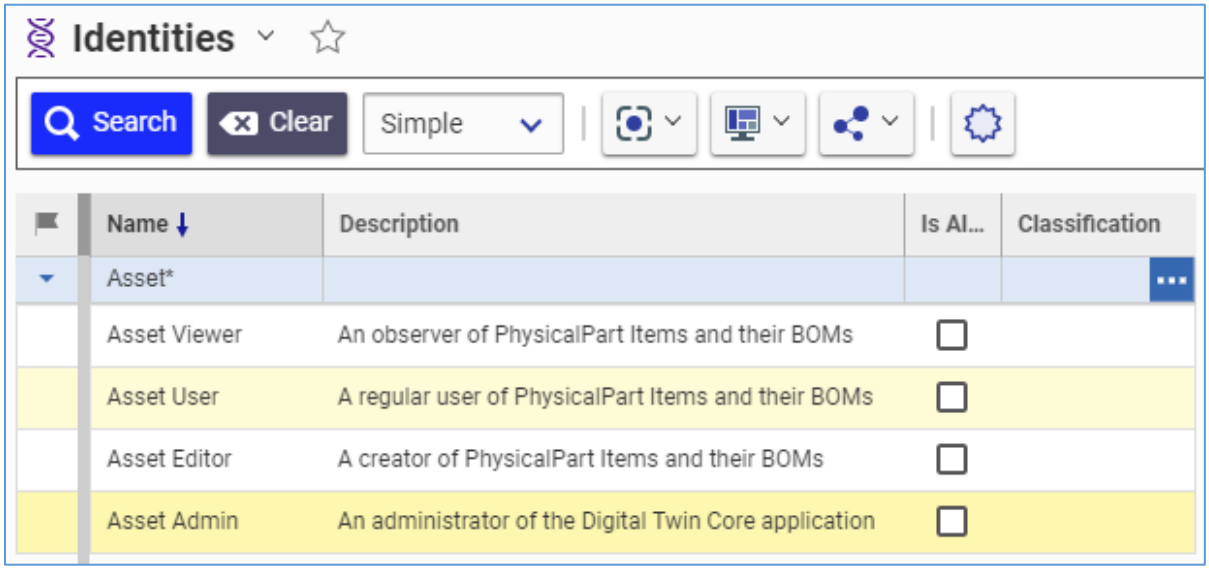



Figure 1.

The Asset Identities are strictly defined to achieve secure control over the Asset Items of the DTC application. Such Items are necessary to represent a real-world asset and its life variables. They are the following:

- **Life Unit**
- **Life Parameter**
- **Life Policy**
- **Part Policy**

- **Physical Part**

As an Aras Innovator Administrator, you manage the Asset Identities: adding and removing members according to business needs using the standard procedures. Out of the box, the Asset Identities are nested as a hierarchy described later in this section.

One non-Asset Identity should belong to one Asset Identity for the DTC control definition.

### 2.1.1 Asset Viewer

The **Asset Viewer** Identity has the view-only Access Rights to the DTC application:

- Has the **Get** and **Can Discover** Access Rights for all DTC Items.

An **Asset Viewer** Identity member can be an inspector who investigates asset and component histories but is not allowed to manipulate assets and components.

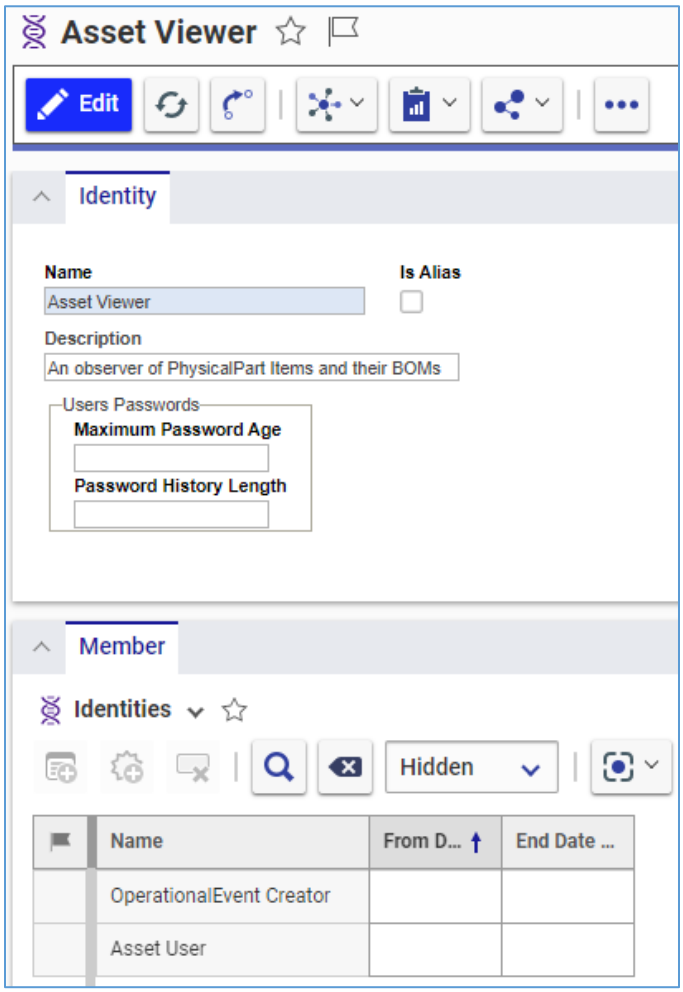



Figure 2.

The **Asset Viewer** Identity has the following members out of the box:

- **Asset User**
- **OperationalEvent Creator**

When creating, editing, or viewing the **Operational Event** Items, members of the OperationalEvent Identities should be able to get, discover, and specify the **Physical Part** and **Life Unit** Items. To obtain the discussed Access Rights, the **OperationalEvent Creator** Identity is added to the **Asset Viewer** Identity.

When not tracking asset life variables and not using OperationalEvent Identities, the **OperationalEvent Creator** Identity can be removed from the **Asset Viewer** Identity.

### 2.1.2 Asset User

The **Asset User** Identity has the following basic Access Rights to the Asset Items:

- Has the same Access Rights as the **Asset Viewer** Identity.
- Can update single-level Physical Part BOMs of the **Physical Part** Items in the **Active** State using the Remove-and-Replace operation.
- Can update editable properties of the **PhysicalPart LifeValue** Relationship Items.
- Can update editable properties of the **PhysicalPart DateValue** Relationship Items.

An **Asset User** Identity member can be an in-the-field technician that repairs, installs, removes, or replaces real-world parts in assemblies and keeps track of the changes done.

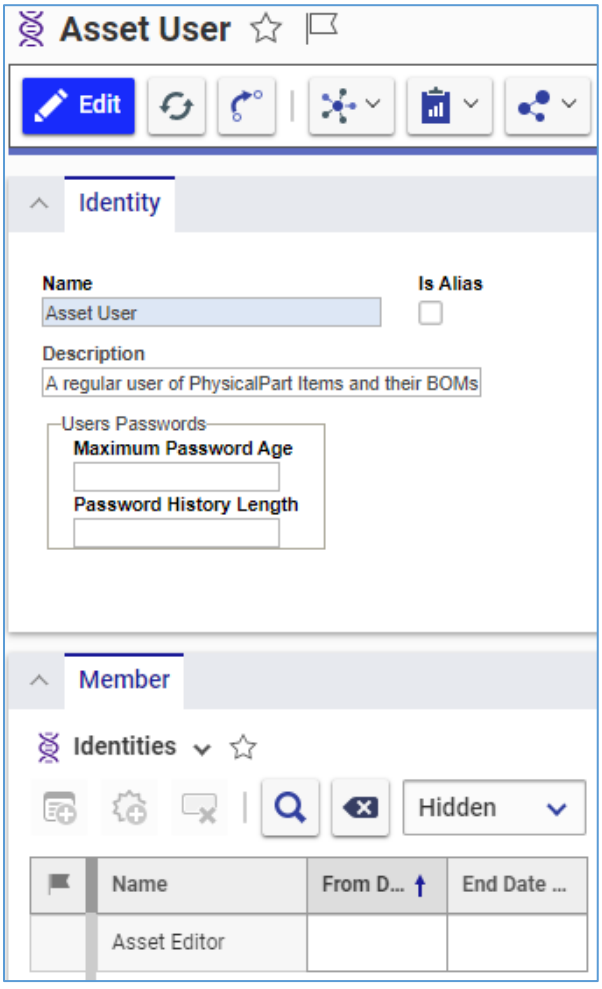



Figure 3.

The **Asset Editor** Identity is the only member of the **Asset User** Identity out of the box.

### 2.1.3 Asset Editor

The **Asset Editor** Identity has the following advanced Access Rights to the Asset Items:

- Has the same Access Rights as the **Asset User** Identity.
- Can create the **Physical Part** Items and BOMs.
- Can promote the **Physical Part** Items.
- Can update the **Physical Part** Items in the **Preliminary** State.
- Can delete the **Physical Part** Items in the **Preliminary** that are not used in the **Operational Event** Items or as BOM components.
- Can create the **Part Policy** Items.
- Can promote the **Part Policy** Items: all to the **Active** State and the unused ones to the **Preliminary** State.
- Can update the **Part Policy** Items in the **Preliminary** State.
- Can delete the unused **Part Policy** Items in the **Preliminary** State.

An **Asset Editor** Identity member can be a workshop supervisor or an officer responsible for digital twins of real-world assets.

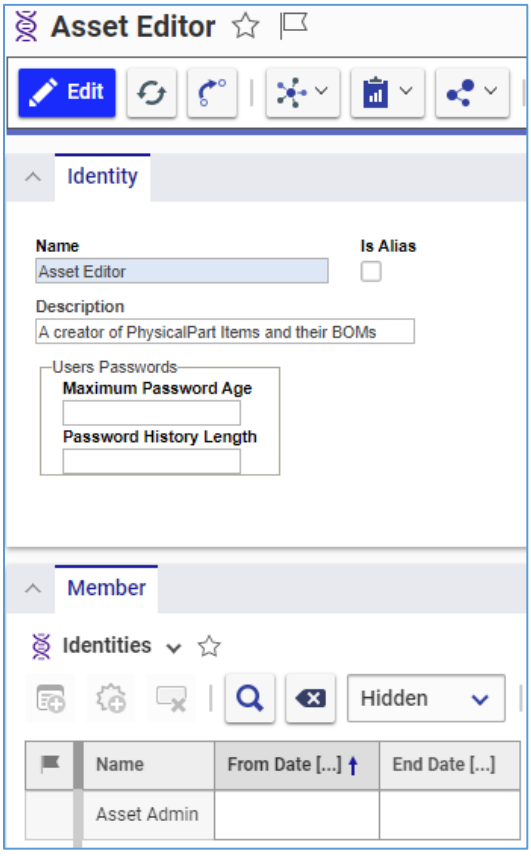



Figure 4.

The **Asset Admin** Identity is the only member of the **Asset Editor** Identity out of the box.

### 2.1.4 Asset Admin

The **Asset Admin** Identity has full Access Rights to the Asset Items:

- Has the same Access Rights as the **Asset Editor** Identity.
- Can update the **Physical Part** Items in any State.
- Has full rights for updating single-level Physical Part BOMs of the source **Physical Part** Items in any State.
- Can delete the **Physical Part** Items in any State that are not used in the **Operational Event** Items or as BOM components.
- Can create the **Life Unit** Items.
- Can delete the unused **Life Unit** Items.
- Can create the **Life Parameter** Items.
- Can promote the **Life Parameter** Items: all to the **Active** State and the unused ones to the **Preliminary** State.
- Can update the **Life Parameter** Items in the **Preliminary** State.
- Can delete the unused **Life Parameter** Items in the **Preliminary** State.
- Can create the **Life Policy** Items.
- Can promote the **Life Policy** Items: all to the **Active** State and the unused ones to the **Preliminary** State.
- Can update the **Life Policy** Items in the **Preliminary** State.
- Can delete the unused **Life Policy** Items in the **Preliminary** State.

An **Asset Admin** Identity member can be a senior executive officer or manager of a maintenance, repair, and overhaul (MRO) department who is aware of and responsible for the consequences that an **Asset Admin** action may produce.

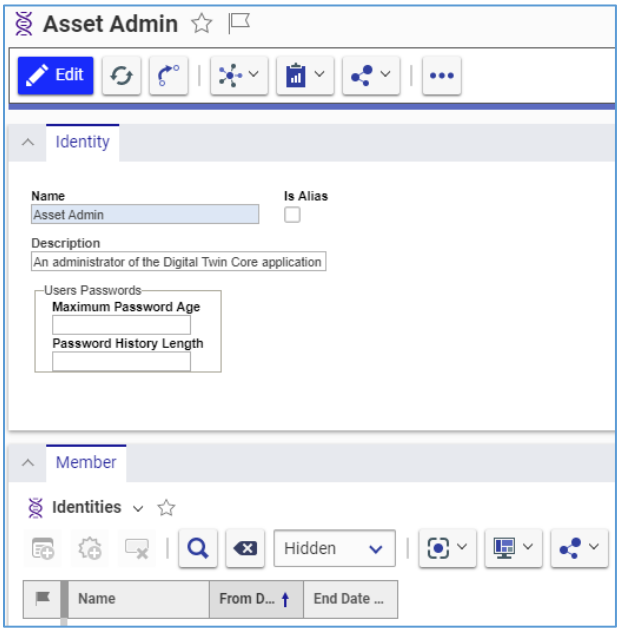



Figure 5.

The **Asset Admin** Identity has no members out of the box.

**Warning** With unlimited Access Rights to the DTC application, an **Asset Admin** member may corrupt the application data. Keep this Identity membership to as few people as possible.

## 2.2 OperationalEvent Identities

To hold specific Access Rights within the Operational Event Permissions for the Operational Items, the DTC application has the following built-in Group Identities that are referred to as the Operational Event Identities:

- **OperationalEvent Creator**—a regular creator and user of the **Operational Event** Items.
- **OperationalEvent Reviewer**—an advanced user of the **Operational Event** Items and a creator of the **Operational Event Type** Items.
- **OperationalEvent Admin**—an administrator of the **Operational Event** and **Operational Event Type** Items.

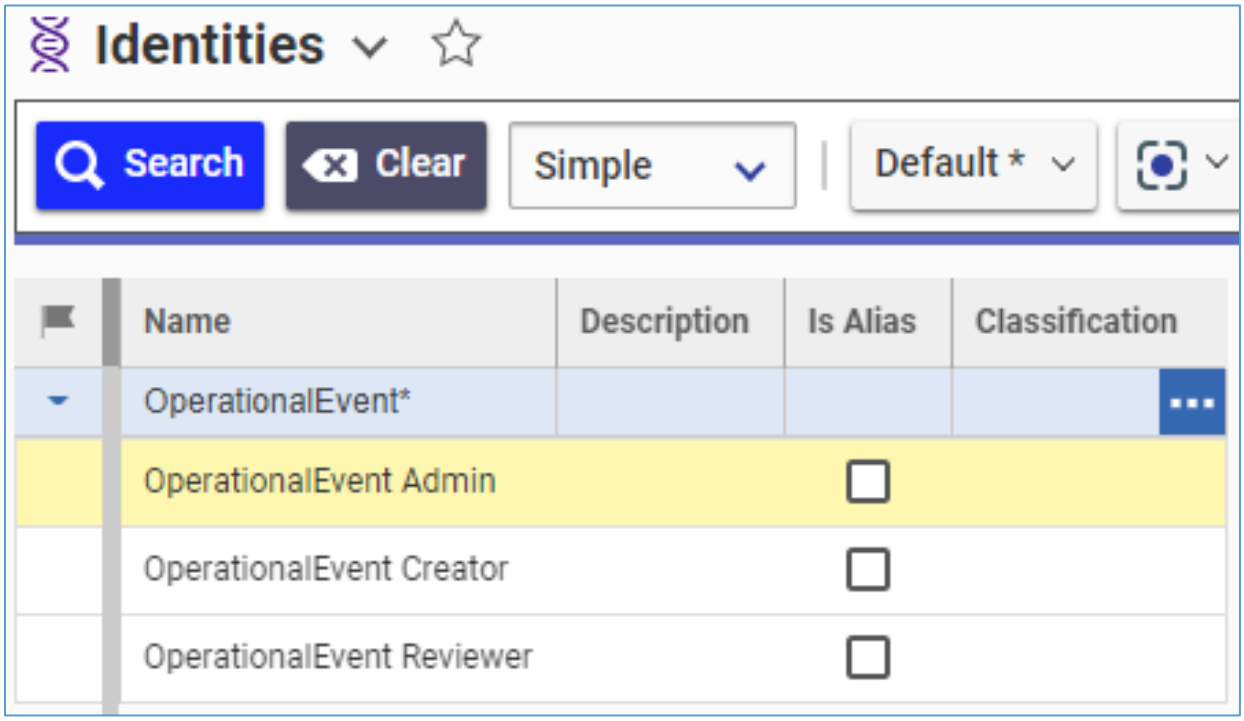



Figure 6.

The Operational Event Identities are strictly defined to achieve secure control over the Operational Items of the DTC application. Such Items are necessary to represent a real-world operational activity event that an asset performs in the field. They are the following:

- **OperationalEvent**
- **OperationalEvent Type**
- **OperationalEvent LifeUnit**

As an Aras Innovator Administrator, you manage the Operational Event Identities: adding and removing members according to business needs using the standard procedures. Out of the box, the Operational Event Identities are nested as a hierarchy described later in this section.

One non-Operational Event Identity should belong to one Operational Event Identity for the DTC control definition.

If your organization does not require tracking asset life variables by its operational events, you can delete the Operational Event Identities from the application.

### 2.2.1 OperationalEvent Creator

The **OperationalEvent Creator** Identity has the following basic Access Rights to the Operational Items:

- Can create the **Operational Event** Items.
- Can promote the **Operational Event** Items to the **Review** State.
- Can update the **Operational Event** Items in the **Preliminary** State.

An **OperationalEvent Creator** Identity member can be an asset operator or technician responsible for logging and tracking the in-the-field operational data of the asset. The **OperationalEvent Creator** Identity creates an **Operational Event** Item and promotes it to the **Review** State for the review by an **OperationalEvent Reviewer** Identity.

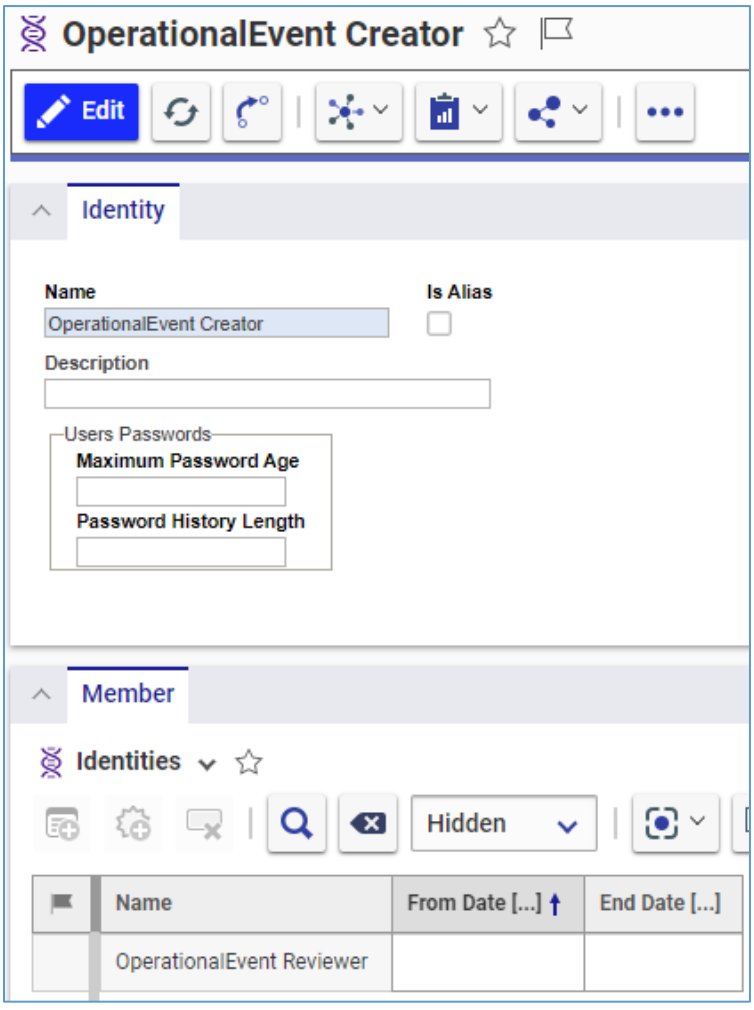



Figure 7.

The **OperationalEvent Reviewer** Identity is the only member of the **OperationalEvent Creator** Identity out of the box.

### 2.2.2 OperationalEvent Reviewer

The **OperationalEvent Reviewer** has the following advanced Access Rights to the Operational Items:

- Has the same Access Rights as the **OperationalEvent Creator** Identity.
- Can update the **Operational Event** Items in the **Review** State.
- Can promote the **Operational Event** Items to the **Preliminary** or **Complete** State.
- Can delete the unused **Operational Event** Items in the **Preliminary** or **Review** State.
- Can create the **Operational Event Type** Items.
- Can update the unused **Operational Event Type** Items.
- Can delete the unused **Operational Event Type** Items.

An **OperationalEvent Reviewer** Identity member can be a shift supervisor or an officer responsible for tracking the operational activities of assets. The **OperationalEvent Reviewer** Identity reviews an **Operational Event** Item in the **Review** State and either approves it by promoting to the **Complete** State or rejects it by promoting back to the **Preliminary** State for correction by an **OperationalEvent Creator** Identity member.

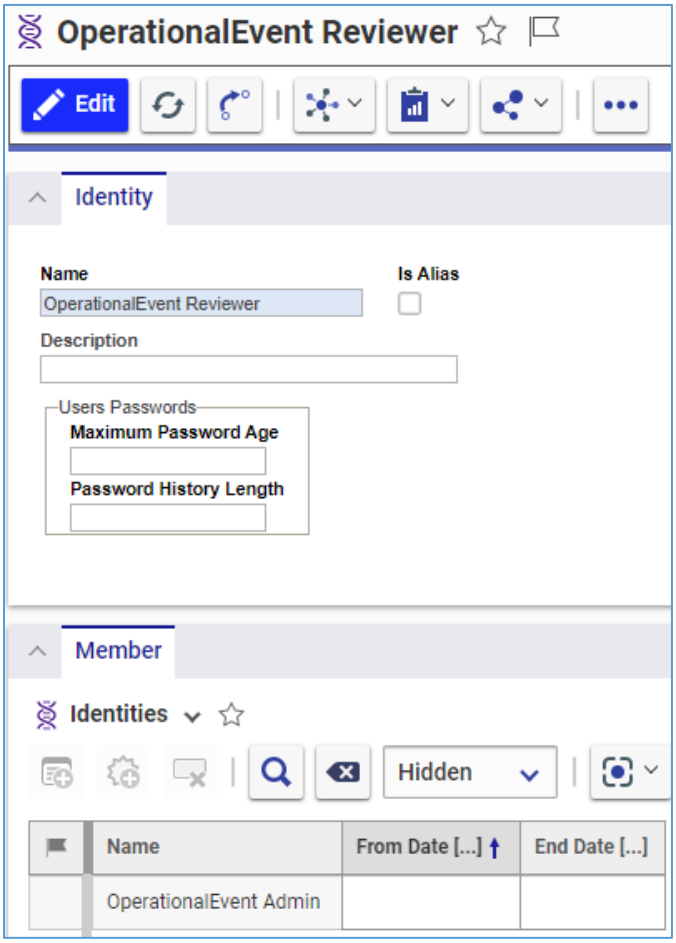



Figure 8.

The **OperationalEvent Admin** Identity is the only member of the **OperationalEvent Reviewer** Identity out of the box.

### 2.2.3 OperationalEvent Admin

The **OperationalEvent Admin** Identity has full Access Rights to the Operational Items:

- Has the same Access Rights as the **OperationalEvent Reviewer** Identity.
- Can update some properties of the **Operational Event** Items in the **Complete** State.

An **OperationalEvent Admin** Identity member can be a senior executive officer or manager of an operational department who is aware of and responsible for the consequences that an **OperationalEvent Admin** action may produce. The **OperationalEvent Admin** Identity is a back door to correct mistakes in the property values of the approved **Operational Event** Items in the **Complete** State.

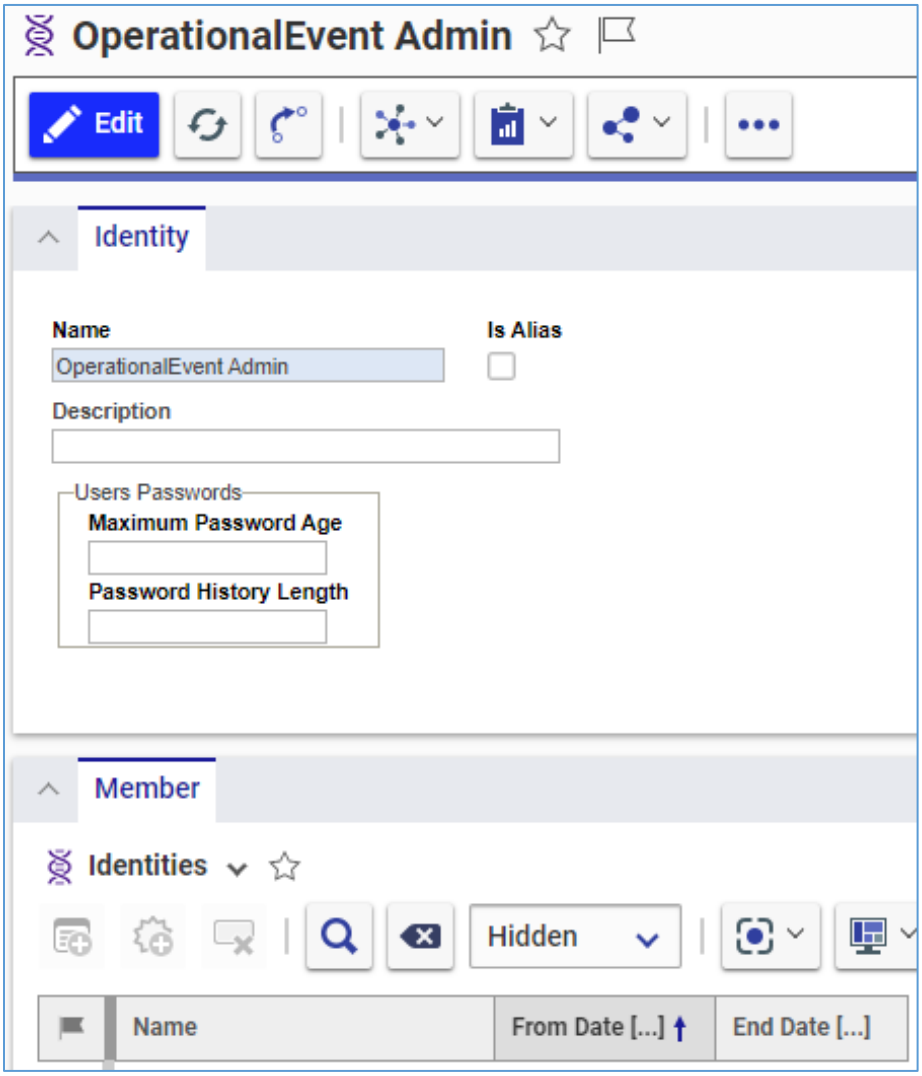



Figure 9.

The **OperationalEvent Admin** Identity has no members out of the box.

## 2.3 Granting Access to the Part Items

All the DTC Identities must have the **Get** and **Can Discover** Access Rights for the **Part** Items that are released, meaning that they are in the **Released** and beyond States: **Released**, **Manual Change**, **In Change**, **Superseded**, and **Obsolete**. Such States have **true** for the **is_released** property, except for **Manual Change** which has **false**.

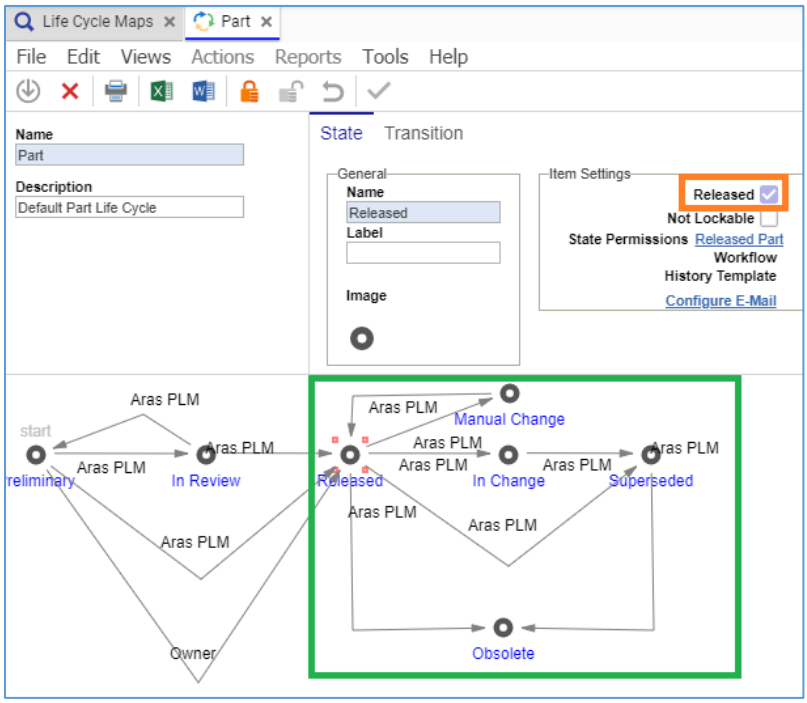



Figure 10.

Having no required access, a DTC Identity does not see foreign **Physical Part** Item properties sourced from **Part** Items. For example, the **Physical Parts** Search Grid displays **Restricted** in the **Part** column.

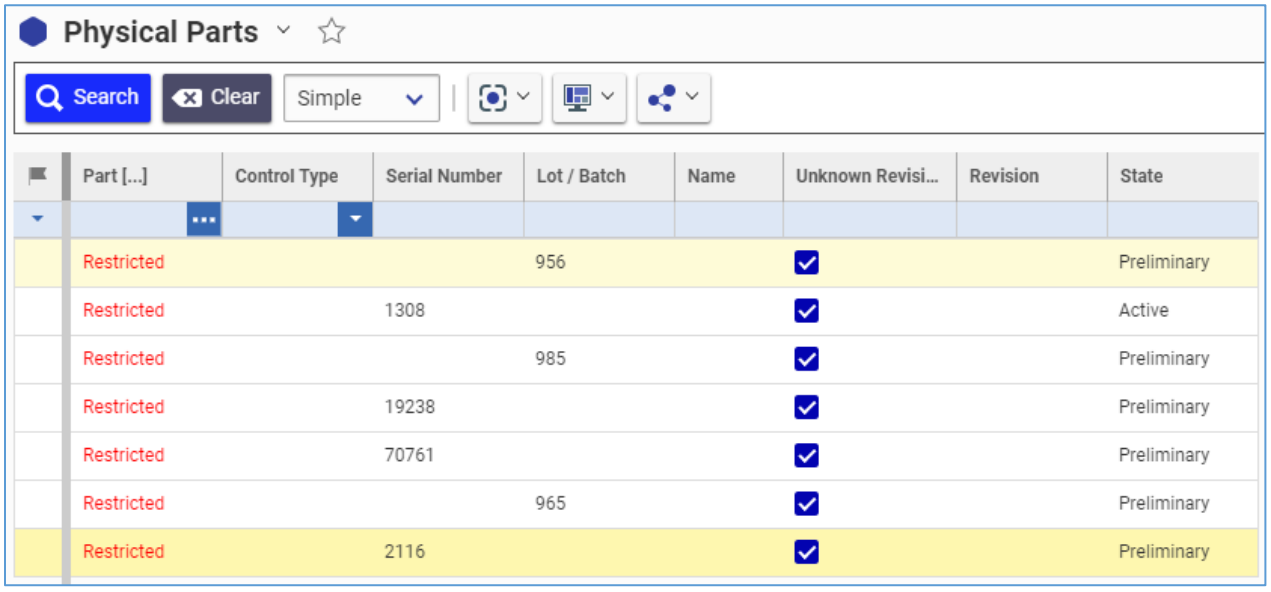



Figure 11.

The **Physical Part** Item view shows **Restricted** correspondingly.

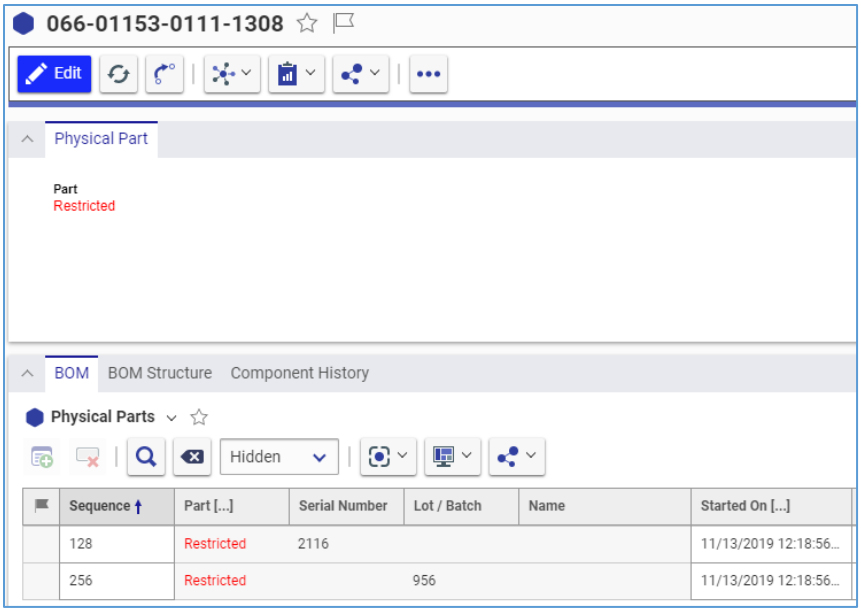



Figure 12.

The **Select Items** dialog will have no **Part** Items for creating a **Physical Part** Item.

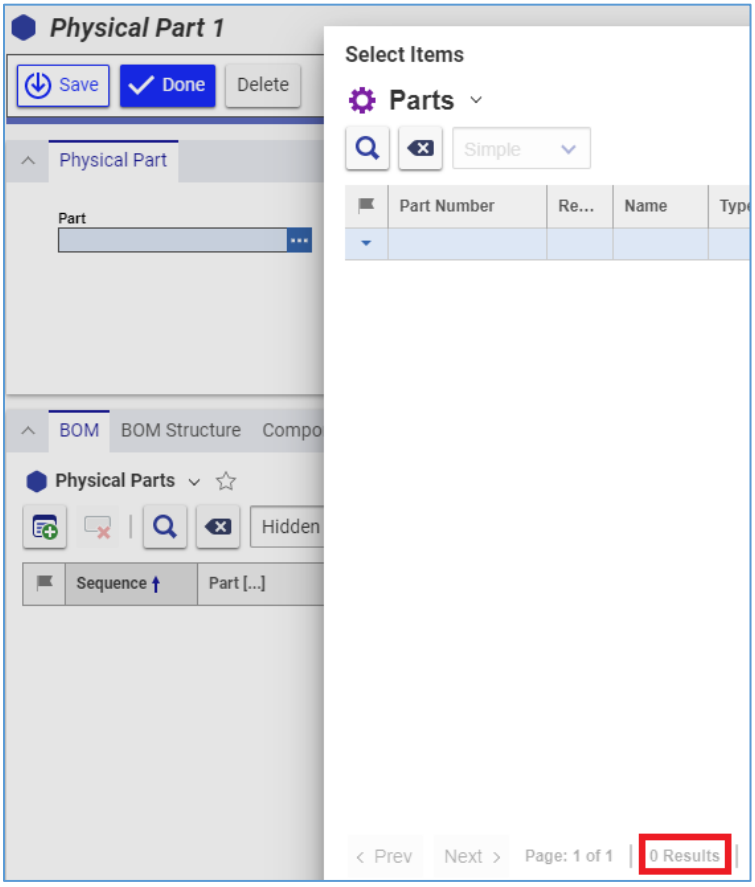



Figure 13.

The out-of-the-box implementation of this requirement is all the Asset Identities must be members of the **All Employees** Identity.

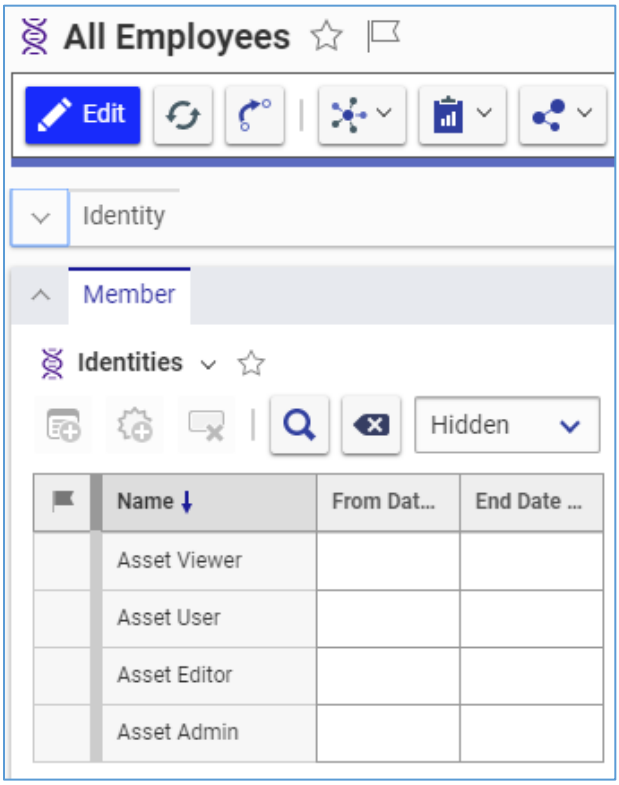



Figure 14.

# 3 Application TOC Access

The **Contents** has the **Assets** category at its top level to access all the DTC application Items.

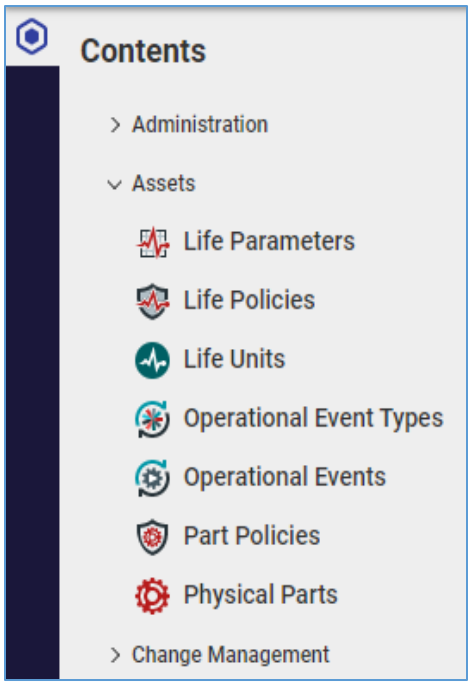



Figure 15.

The **Assets** category is visible for the **World** Identity out of the box. This visibility is set in the **PhysicalPart** ItemType as follows:

1. The **PhysicalPart** ItemType TOC Access is set to the **Asset** Category for the **World** Identity.

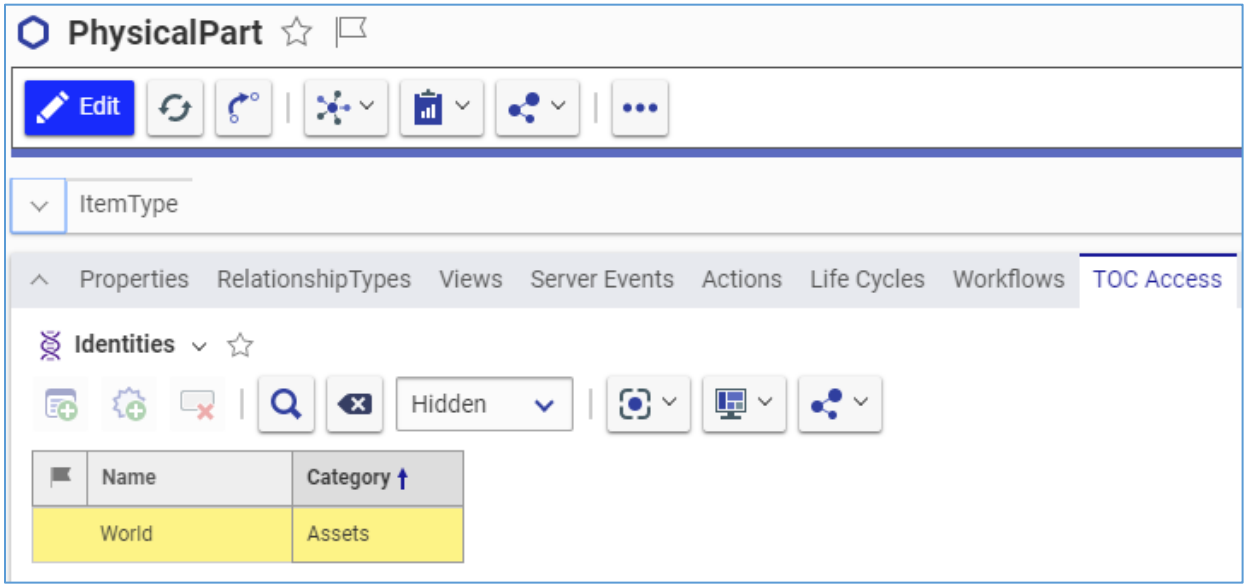



Figure 16.

2. The **PhysicalPart** ItemType Client Style is set to the **PhysicalPart** Presentation Configuration.

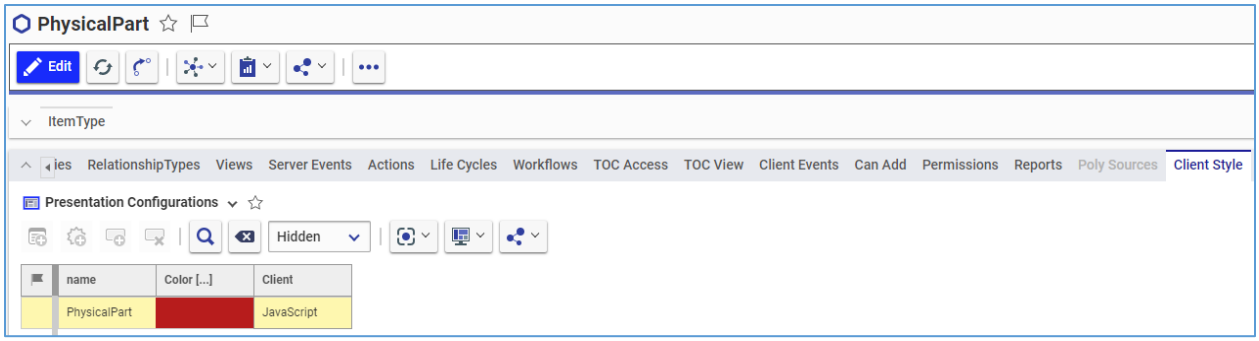



Figure 17.

3. The **PhysicalPart** Presentation Configuration is visible to **World** in **TOC**.

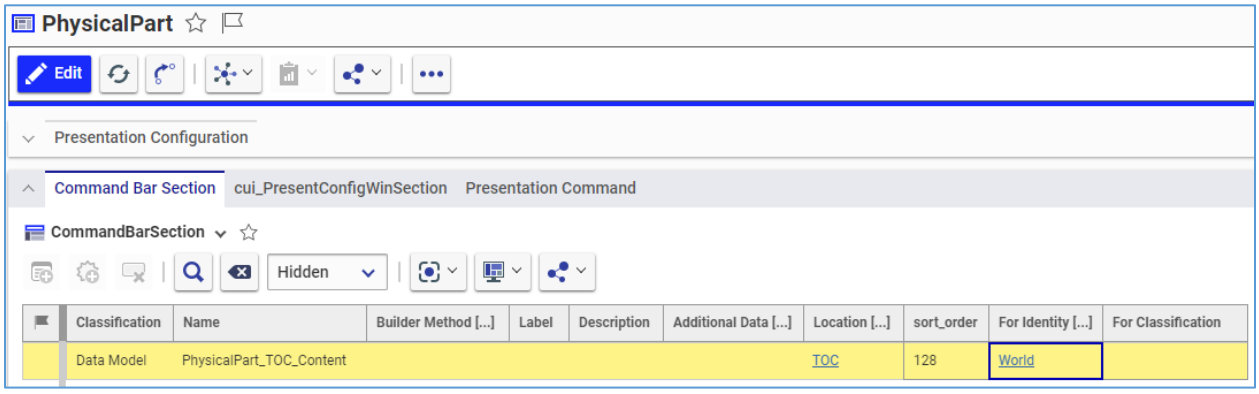



Figure 18.

If a member of the **World** Identity does not belong to an Asset Identity, this member can access the **Physical Parts** TOC entry and Search Grid but cannot see DTC Items.

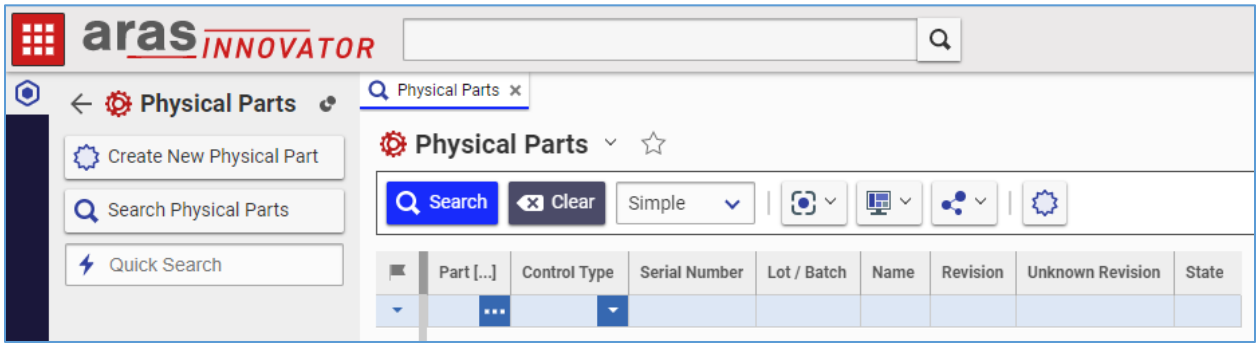



Figure 19.

Once the DTC application is installed, you should configure the **Assets** category visibility according to business needs.

# 4 DTC Item Versions

None of the ItemTypes in the DTC application are versionable. Each of them has its **Versionable** property set to **false**. The system does not store a history of DTC Item properties. Enabling versioning for a DTC ItemType must be avoided as it may decrease application performance, including the output of large amounts of data.

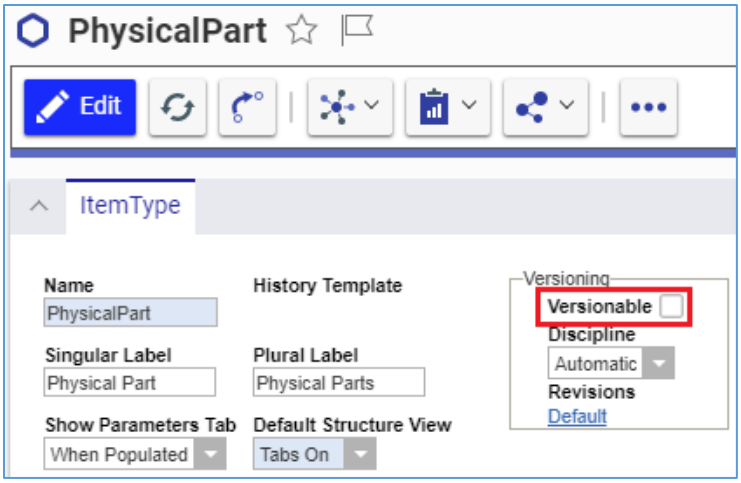



Figure 20.

# 5 DTC Relationship sort_order Customization

## 5.1 The php_setDefaultSequenceNumber Method

The **php_setDefaultSequenceNumber** Method of a Relationship Grid Event customizes the default automatic **Sequence** (**sort_order**) numbering in the Relationship Grid of a DTC Item by assigning multiples of ten.

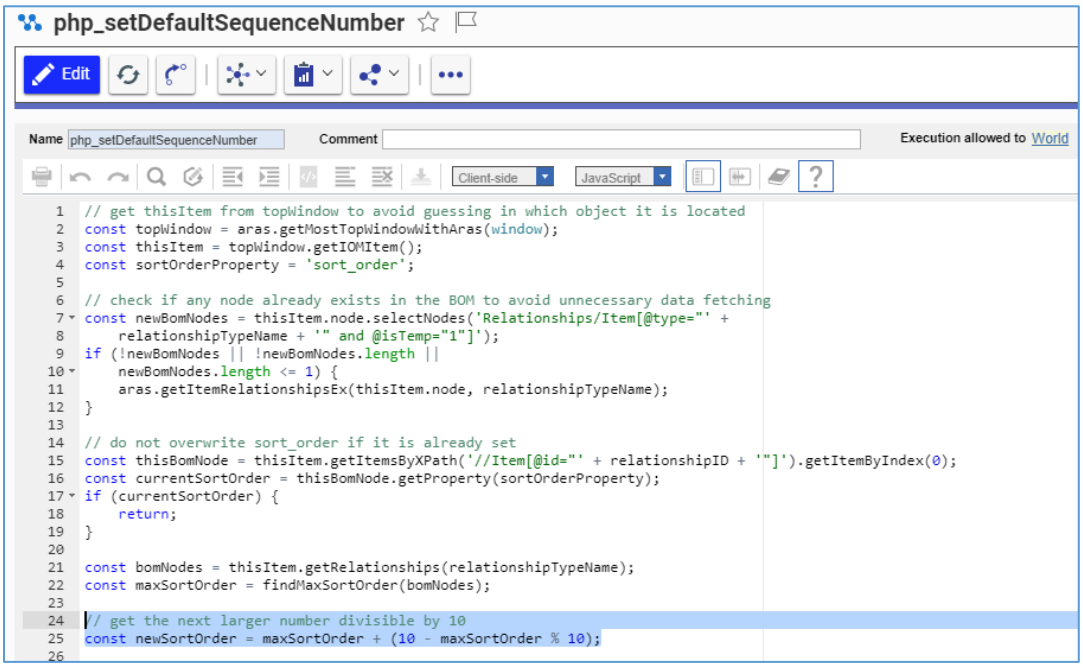



Figure 21.

Each DTC Item in the Grid has its **Sequence** value automatically set to a divided-by-ten number that goes next after the highest previous **Sequence** value. For example, if there are Items with **Sequence** values such as 10, 20, 24, 30, 39, the next one will the value 40; if 12, 32, 41, 55, 63, it will be 70; if 10, 28, 33, 27, it will be 40.

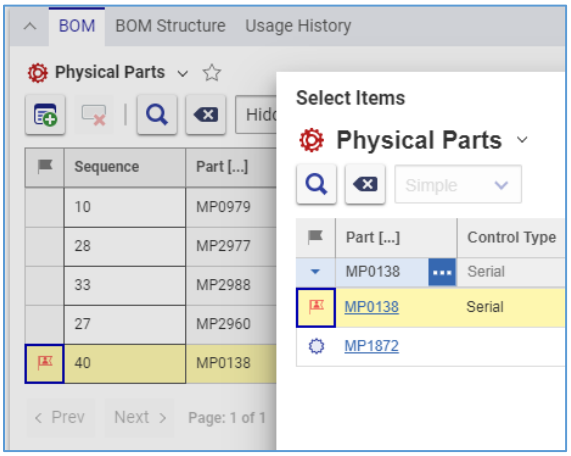



Figure 22.

The **php_setDefaultSequenceNumber** Method works only on the client side. When creating DTC Relationship Items with AML, their **sort_order** numbering must follow the Method numbering logic. Otherwise, these DTC Relationship Items will have **Sequence** values assigned as multiples of 128 (default logic). The DTC Relationship structures also will have Items with **Sequence** values set with both types of logic if a user adds Items using the UI.

The numbering customization coincides with common industry practice. It also makes digital twin representations of large assemblies, like aircraft or vessels, easier because they may have thousands of monitored components. The default incrementation (128, 256, 384, and so on) might produce enormously large numbers.

## 5.2 DTC Relationship Item sort_order Numbering with AML

The DTC application customizes the default Innovator logic of assigning `sort_order` values to new Relationship Items.

Because the custom **php_setDefaultSequenceNumber** Method works only on the client side, you must explicitly set up the `sort_order` values following the custom logic when creating **Physical Part BOM** Items via AML.

```
<AML>
   <Item type="PhysicalPart BOM" action="add" doGetItem="0">
      <source_id>ABC2956084B7441285F10F070BD8A62F</source_id>
      <related_id>ABC2956084B7441285F10F070BD8A84A</related_id>
      <sort_order>40</sort_order>
      <reference_designator>Example</reference_designator>
      <start_on>2020-04-19T00:00:00</start_on>
   </Item>
</AML>
```

Violating this requirement may mix up the ordering of the `sort_order` values for sorting purposes.

If a DTC Item is added with AML after DTC Items have already been created with the GUI, the AML DTC Relationship Item will have its `sort_order` value as a sum of 128 and the highest current `sort_order` value.

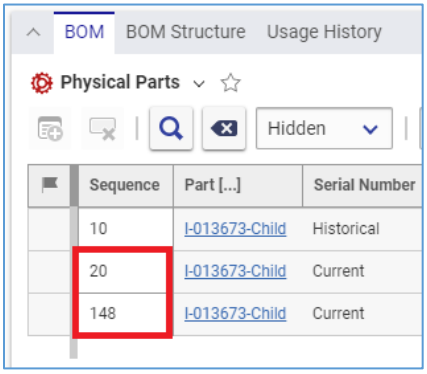



Figure 23.

If a DTC Item is added with the GUI after DTC Items have already been created with AML, the GUI DTC Relationship Item will have its `sort_order` value as the next ten-divisible number after the highest current `sort_order` value.

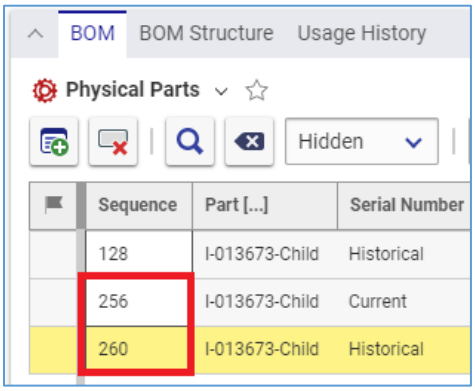



Figure 24.

# 6 Physical Part Items

## 6.1 Physical Part Item Overview

A **Physical Part** Item is an Item of the **PhysicalPart** ItemType.

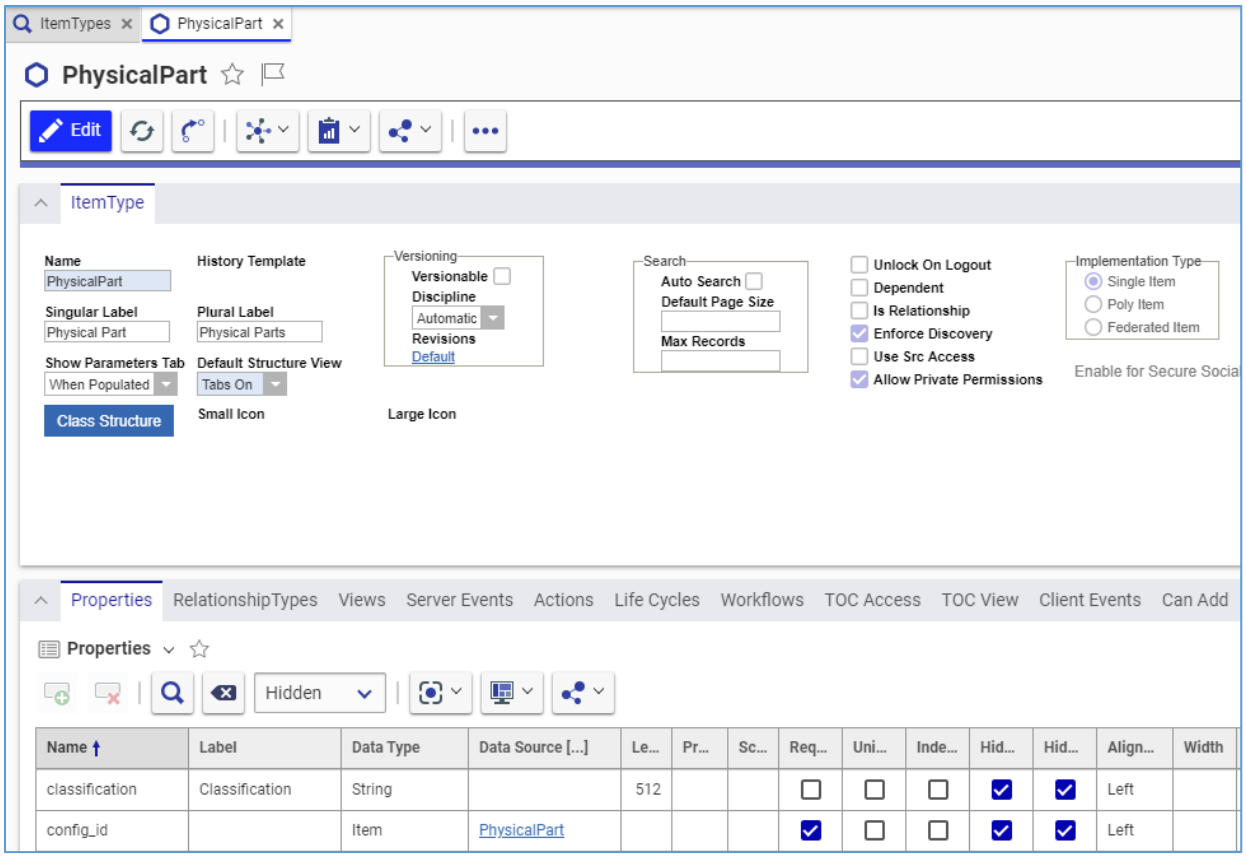



Figure 25.

A **Physical Part** Item should have its **Part Number** and **Revision**, inventory ID (**Serial** or **Lot Number**) unchanged once they are assigned. The **Part Number** and **Revision** identify the given **Physical Part** against other **Physical Parts** in the system as a series of similar **Physical Parts**, while the inventory ID—against **Physical Parts** has the same **Part Number**. Changing any of these properties is redefining the **Physical Part** Item, which is not possible for a represented real-world item. In case of an error in a property, an **Asset Admin** has Access Rights to correct it.

## 6.2 Physical Part Permissions

The default Permission for the **Physical Part** Items is **php_DefaultPhysicalPart**.

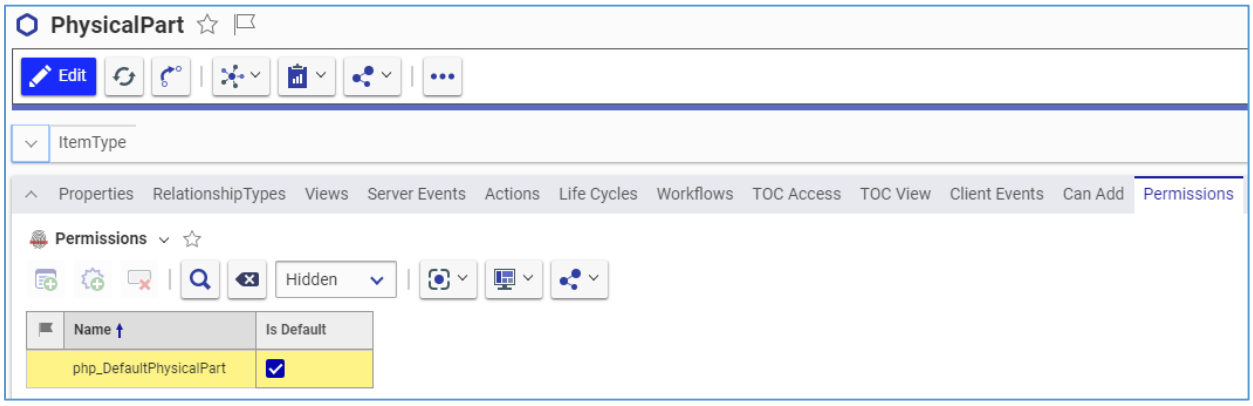



Figure 26.

It is the strictest among the **Physical Part** Permissions granting only **Get** and **Can Discover** Access Rights to all the Asset Identities.

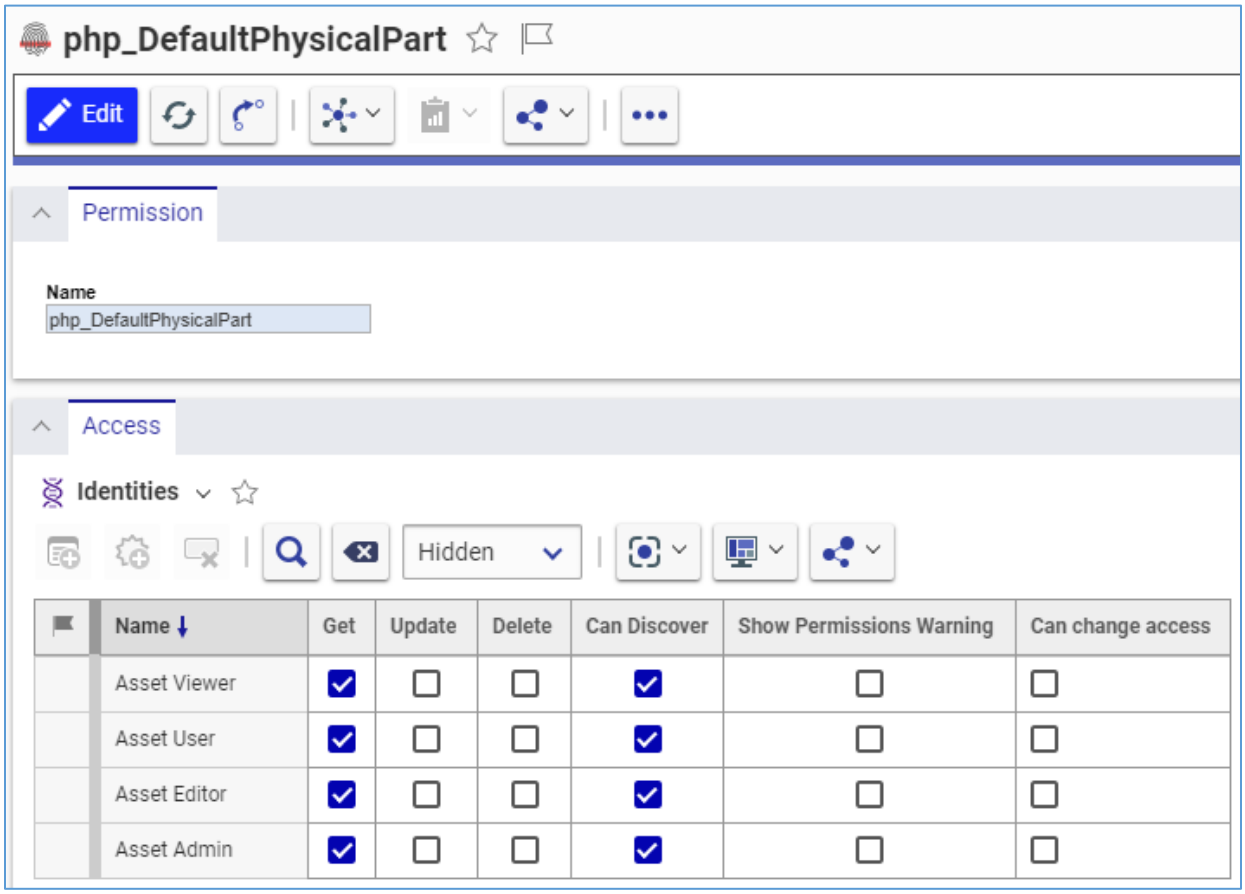



Figure 27.

The **PhysicalPart** Life Cycle Map overrides the default **php_DefaultPhysicalPart** Permission by setting another **Physical Part** Permission in a particular Life Cycle State for a **Physical Part** Item. A newly

created **Physical Part** Item is in the **Preliminary** state where it has the **php_PreliminaryPhysicalPart** Permission assigned, not the default one.

The following table describes the **PhysicalPart** Life Cycle Permissions. The **Promoted by** column shows the Identities that can promote an Item to a given Life Cycle State. The **Permission** column refers to a Permission that manages Access Rights in a given Life Cycle State.

Table 1: The Physical Part Permissions matrix

| Life Cycle State | Can Add | Update | Delete | Change access | Promoted by | Permission |
|---|---|---|---|---|---|---|
| Preliminary | Asset Editor, Asset Admin | Asset Editor, Asset Admin | Asset Editor, Asset Admin | Asset Admin | NA | php_PreliminaryPhysicalPart |
| Active | NA | Asset User, Asset Editor, Asset Admin | Asset Admin | Asset Admin | Asset Editor, Asset Admin | php_ActivePhysicalPart |

**Note:** All the Asset Identities, including **Asset Viewer**, have the **Get** and **Can Discover** Access Rights for the **Physical Part** Items in any Item State. The **Show Permissions Warning** Access Rights are granted to no one.

## 6.3 Foreign PhysicalPart Item Properties and AML

Some of the **PhysicalPart** Item properties are foreign: they are sourced from a related **Part** Item. For details on these properties, refer to the *Aras Digital Twin Core 12.0R2 – User Guide*.

The DTC application handles **PhysicalPart** Item AML requests with the given foreign properties differently depending on the request type and property.

### 6.3.1 PhysicalPart skipValidation attribute

By default, the AML subqueries do not work for the **PhysicalPart** Items because of enabled validation.

The DTC application includes the `skipValidation` attribute for the AML requests, which disables the validation and enables the subqueries. This attribute works only for one given **PhysicalPart** Item and when connected as a member of the **Asset Admin** Identity.

You must be sure about the validity of the **PhysicalPart** Item data when applying this attribute because it turns off the default application validation for the given **PhysicalPart** Item.

For example, set `skipValidation` to `1` to enable subquerying for a **Part** Item within a single **PhysicalPart** Item AML request.

### 6.3.2 PhysicalPart Add Requests with Subqueries

By default, an `add` request works if the required `part` property is an **`ID`** of a related **Part** Item.

```
<AML>
  <Item type='PhysicalPart' action='add' doGetItem='0'>
    <part>A21521ED0D2648228757C77E9FE55625</part>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Result>
      <Item type="PhysicalPart" id="737452E1AC9041B7865204532529A796" />
    </Result>
    <Message>
      <event name="ids_modified" value="737452E1AC9041B7865204532529A796" />
    </Message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 28.

An error is raised against the `add` request by default if the `part` property value is a **Part Number** or a subquery for this **Part** Item.

```
<AML>
  <Item type='PhysicalPart' action='add' doGetItem='0'>
    <part>
      <Item type='Part' action='get' select='id'>
        <item_number>I-013673-Parent</item_number>
      </Item>
    </part>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n="http://www.aras.com/I18N">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>1</faultcode>
      <faultactor />
      <faultstring>The "Part" property is required. Provide a value for this property before saving.</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 29.

**Note:** "*Only Released Part Items are allowed for creating Physical Parts.*" is the `faultstring` if a `part` property value is a **`Part Number`** because the system takes any string value given in this property as an ID and does not find this ID among released **Part** Items. This `faultstring` is the same for released and not released **Part** Items in this case.

```
<AML>
  <Item type='PhysicalPart' action='add' doGetItem='0'>
    <part>I-013673-Parent</part>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n="http://www.aras.com/I18N">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>1</faultcode>
      <faultactor />
      <faultstring>Only Released Part Items are allowed for creating Physical Parts.</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 30.

An `add` request works for a single **PhysicalPart** Item if the following criteria is met:

- Connected as a member of the **Asset Admin** Identity.
- The `skipValidation` attribute is set to `1`.
- The `part` property is a subquery for a related **Part** Item.

**Warning** Make sure that given **Part** and **PhysicalPart** Item data is valid when applying the `skipValidation` attribute as it turns off the validation and may allow corrupted data, such as a **PhysicalPart** Item related to a **Part** Item that is not released.

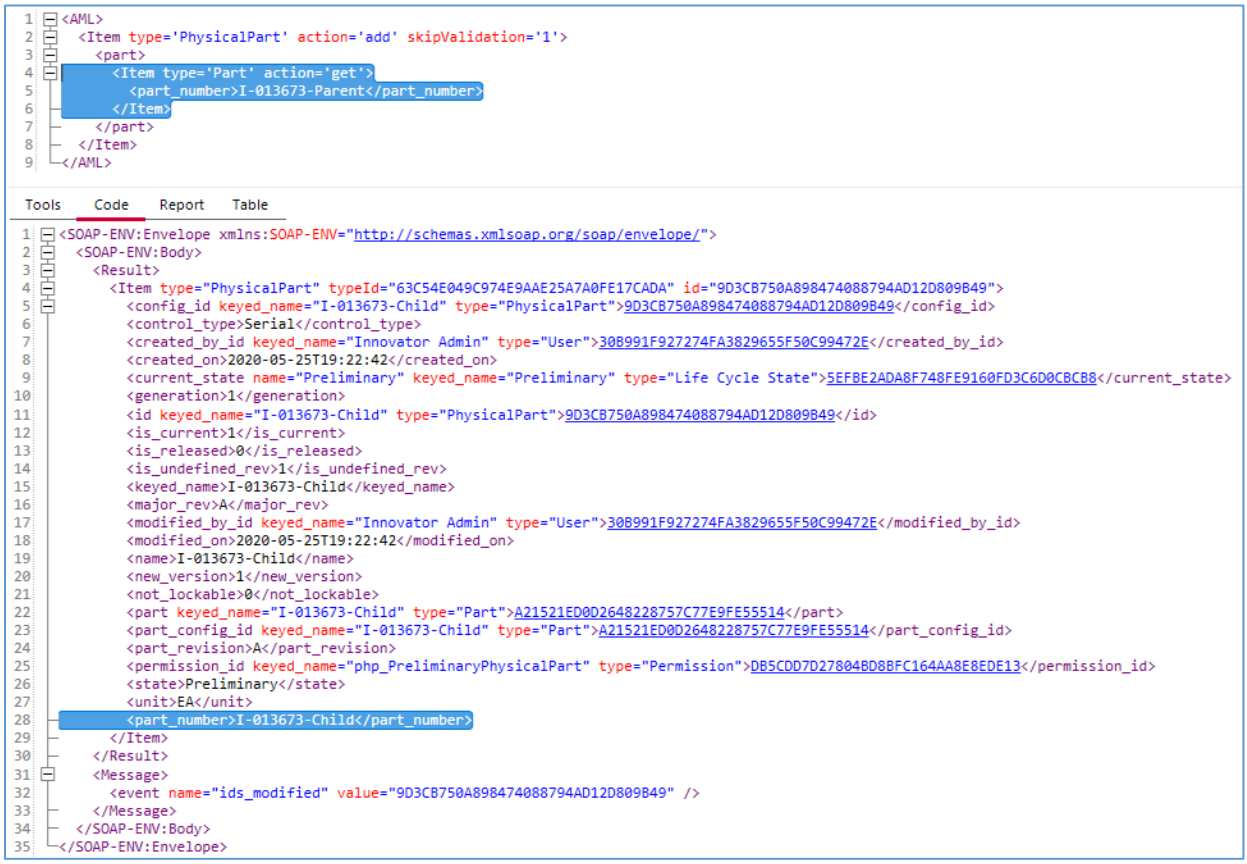
```
<AML>
  <Item type='PhysicalPart' action='add' skipValidation='1'>
    <part>
      <Item type='Part' action='get' select='id'>
        <item_number>I-013673-Parent</item_number>
      </Item>
    </part>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Result>
      <Item type="PhysicalPart" typeId="63C54E049C974E9AAE25A7A0FE17CADA" id="B5993D89B5814798A6CB4E36902175B2">
        <config_id keyed_name="I-013673-Parent" type="PhysicalPart">B5993D89B5814798A6CB4E36902175B2</config_id>
        <control_type>Serial</control_type>
        <created_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</created_by_id>
        <created_on>2020-05-25T19:42:25</created_on>
        <current_state name="Preliminary" keyed_name="Preliminary" type="Life Cycle State">5EFBE2ADA8F748FE9160FD3C6D0CBCB8</current_state>
        <generation>1</generation>
        <id keyed_name="I-013673-Parent" type="PhysicalPart">B5993D89B5814798A6CB4E36902175B2</id>
        <is_current>1</is_current>
        <is_released>0</is_released>
        <is_undefined_rev>1</is_undefined_rev>
        <keyed_name>I-013673-Parent</keyed_name>
        <major_rev>A</major_rev>
        <modified_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</modified_by_id>
        <modified_on>2020-05-25T19:42:25</modified_on>
        <name>I-013673-Parent</name>
        <new_version>1</new_version>
        <not_lockable>0</not_lockable>
        <part keyed_name="I-013673-Parent" type="Part">A21521ED0D2648228757C77E9FE55625</part>
        <part_config_id keyed_name="I-013673-Parent" type="Part">A21521ED0D2648228757C77E9FE55625</part_config_id>
        <part_revision>A</part_revision>
        <permission_id keyed_name="php_PreliminaryPhysicalPart" type="Permission">DB5CDD7D27804BD8BFC164AA8E8EDE13</permission_id>
        <state>Preliminary</state>
        <unit>EA</unit>
        <part_number>I-013673-Parent</part_number>
      </Item>
    </Result>
    <Message>
      <event name="ids_modified" value="B5993D89B5814798A6CB4E36902175B2" />
    </Message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 31.

**Note:** The **Part Number** property is the `item_number` for a **Part** Item but the `part_number` for a **PhysicalPart** Item. If misidentified as the `part_number` given for a **Part** Item, a new **PhysicalPart** Item will be created but related to another, random **Part** Item.




```
<AML>
  <Item type='PhysicalPart' action='add' skipValidation='1'>
    <part>
      <Item type='Part' action='get'>
        <part_number>I-013673-Parent</part_number>
      </Item>
    </part>
  </Item>
</AML>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Result>
      <Item type="PhysicalPart" typeId="63C54E049C974E9AAE25A7A0FE17CADA" id="9D3CB750A898474088794AD12D809B49">
        <config_id keyed_name="I-013673-Child" type="PhysicalPart">9D3CB750A898474088794AD12D809B49</config_id>
        <control_type>Serial</control_type>
        <created_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</created_by_id>
        <created_on>2020-05-25T19:22:42</created_on>
        <current_state name="Preliminary" keyed_name="Preliminary" type="Life Cycle State">5EFBE2ADA8F748FE9160FD3C6D0CBCB8</current_state>
        <generation>1</generation>
        <id keyed_name="I-013673-Child" type="PhysicalPart">9D3CB750A898474088794AD12D809B49</id>
        <is_current>1</is_current>
        <is_released>0</is_released>
        <is_undefined_rev>1</is_undefined_rev>
        <keyed_name>I-013673-Child</keyed_name>
        <major_rev>A</major_rev>
        <modified_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</modified_by_id>
        <modified_on>2020-05-25T19:22:42</modified_on>
        <name>I-013673-Child</name>
        <new_version>1</new_version>
        <not_lockable>0</not_lockable>
        <part keyed_name="I-013673-Child" type="Part">A21521ED0D2648228757C77E9FE55514</part>
        <part_config_id keyed_name="I-013673-Child" type="Part">A21521ED0D2648228757C77E9FE55514</part_config_id>
        <part_revision>A</part_revision>
        <permission_id keyed_name="php_PreliminaryPhysicalPart" type="Permission">DB5CDD7D27804BD8BFC164AA8E8EDE13</permission_id>
        <state>Preliminary</state>
        <unit>EA</unit>
        <part_number>I-013673-Child</part_number>
      </Item>
    </Result>
    <Message>
      <event name="ids_modified" value="9D3CB750A898474088794AD12D809B49" />
    </Message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 32.

### 6.3.3 PhysicalPart get Requests with Foreign Properties

By default, a `get` request works if you are specifying the necessary **PhysicalPart** Items with a foreign property.

```
<AML>
  <Item type='PhysicalPart' action='get'>
    <name>I-013673-Parent</name>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Result>
      <Item type="PhysicalPart" typeId="63C54E049C974E9AAE25A7A0FE17CADA" id="CFD2956084B7441285F10F070BD8A62F">
        <config_id keyed_name="I-013673-Parent Example" type="PhysicalPart">CFD2956084B7441285F10F070BD8A62F</config_id>
        <control_type>Serial</control_type>
        <created_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</created_by_id>
        <created_on>2020-05-21T12:05:20</created_on>
        <current_state name="Active" keyed_name="Active" type="Life Cycle State">0D0932D95A7E47229044E6632BA610E3</current_state>
        <generation>1</generation>
        <id keyed_name="I-013673-Parent Example" type="PhysicalPart">CFD2956084B7441285F10F070BD8A62F</id>
        <is_current>1</is_current>
        <is_released>0</is_released>
        <is_undefined_rev>1</is_undefined_rev>
        <keyed_name>I-013673-Parent Example</keyed_name>
        <lot_number></lot_number>
        <major_rev>A</major_rev>
        <modified_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</modified_by_id>
        <modified_on>2020-05-22T17:15:05</modified_on>
        <name>I-013673-Parent</name>
        <new_version>1</new_version>
        <not_lockable>0</not_lockable>
        <part keyed_name="I-013673-Parent" type="Part">A21521ED0D2648228757C77E9FE55625</part>
        <part_config_id keyed_name="I-013673-Parent" type="Part">A21521ED0D2648228757C77E9FE55625</part_config_id>
        <part_revision>A</part_revision>
        <permission_id keyed_name="php_ActivePhysicalPart" type="Permission">8513E67730BA4A4EA7AB782CF7443786</permission_id>
        <serial_number>Example</serial_number>
        <state>Active</state>
        <unit>EA</unit>
        <part_number>I-013673-Parent</part_number>
      </Item>
    </Result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 33.

The `part` property should be an **`ID`** of a related **Part** Item or a subquery for this **Part** Item in the `get` request by default.

```
<AML>
  <Item type='PhysicalPart' action='get'>
    <part>
      <Item type='Part'>
        <item_number>I-013673-Parent</item_number>
      </Item>
    </part>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Result>
      <Item type="PhysicalPart" typeId="63C54E049C974E9AAE25A7A0FE17CADA" id="CFD2956084B7441285F10F070BD8A62F">
        <config_id keyed_name="I-013673-Parent Example" type="PhysicalPart">CFD2956084B7441285F10F070BD8A62F</config_id>
        <control_type>Serial</control_type>
        <created_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</created_by_id>
        <created_on>2020-05-21T12:05:20</created_on>
        <current_state name="Active" keyed_name="Active" type="Life Cycle State">0D0932D95A7E47229044E6632BA610E3</current_state>
        <generation>1</generation>
        <id keyed_name="I-013673-Parent Example" type="PhysicalPart">CFD2956084B7441285F10F070BD8A62F</id>
        <is_current>1</is_current>
        <is_released>0</is_released>
        <is_undefined_rev>1</is_undefined_rev>
        <keyed_name>I-013673-Parent Example</keyed_name>
        <lot_number></lot_number>
        <major_rev>A</major_rev>
        <modified_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</modified_by_id>
        <modified_on>2020-05-22T17:15:05</modified_on>
        <name>I-013673-Parent</name>
        <new_version>1</new_version>
        <not_lockable>0</not_lockable>
        <part keyed_name="I-013673-Parent" type="Part">A21521ED0D2648228757C77E9FE55625</part>
        <part_config_id keyed_name="I-013673-Parent" type="Part">A21521ED0D2648228757C77E9FE55625</part_config_id>
        <part_revision>A</part_revision>
        <permission_id keyed_name="php_ActivePhysicalPart" type="Permission">8513E67730BA4A4EA7AB782CF7443786</permission_id>
        <serial_number>Example</serial_number>
        <state>Active</state>
        <unit>EA</unit>
        <part_number>I-013673-Parent</part_number>
      </Item>
    </Result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 34.

An error is raised if the `part` property value is a **`Part Number`** in the `get` request.

```
<AML>
  <Item type="PhysicalPart" action="get">
    <part>I-013673-Parent</part>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault xmlns:af="http://www.aras.com/InnovatorFault">
      <faultcode>0</faultcode>
      <faultstring><![CDATA[No items of type PhysicalPart found.]]></faultstring>
      <detail>
        <af:legacy_detail><![CDATA[No items of type PhysicalPart found.]]></af:legacy_detail>
        <af:legacy_faultstring><![CDATA[No items of type 'PhysicalPart' found using the criteria:
<Item type="PhysicalPart" action="get">
  <part>I-013673-Parent</part>
  </Item>
]]></af:legacy_faultstring>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 35.

### 6.3.4 PhysicalPart Item Edit Requests with Foreign Properties

An `edit` request works if the `part` property is an **`ID`** of a related **Part** Item in the `where` attribute clause.

```
<AML>
  <Item type='PhysicalPart' action='edit' where="[PhysicalPart].part='A21521ED0D2648228757C77E9FE55625'">
  <serial_number>Edited</serial_number>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Result>
      <Item type="PhysicalPart" typeId="63C54E049C974E9AAE25A7A0FE17CADA" id="CFD2956084B7441285F10F070BD8A62F">
        <config_id keyed_name="I-013673-Parent Edited" type="PhysicalPart">CFD2956084B7441285F10F070BD8A62F</config_id>
        <control_type>Serial</control_type>
        <created_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</created_by_id>
        <created_on>2020-05-21T12:05:20</created_on>
        <current_state name="Active" keyed_name="Active" type="Life Cycle State">0D0932D95A7E47229044E6632BA610E3</current_state>
        <generation>1</generation>
        <id keyed_name="I-013673-Parent Edited" type="PhysicalPart">CFD2956084B7441285F10F070BD8A62F</id>
        <is_current>1</is_current>
        <is_released>0</is_released>
        <is_undefined_rev>1</is_undefined_rev>
        <keyed_name>I-013673-Parent Edited</keyed_name>
        <lot_number></lot_number>
        <major_rev>A</major_rev>
        <modified_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</modified_by_id>
        <modified_on>2020-05-23T18:39:58</modified_on>
        <name>I-013673-Parent</name>
        <new_version>1</new_version>
        <not_lockable>0</not_lockable>
        <part keyed_name="I-013673-Parent" type="Part">A21521ED0D2648228757C77E9FE55625</part>
        <part_config_id keyed_name="I-013673-Parent" type="Part">A21521ED0D2648228757C77E9FE55625</part_config_id>
        <part_revision>A</part_revision>
        <permission_id keyed_name="php_ActivePhysicalPart" type="Permission">8513E67730BA4A4EA7AB782CF7443786</permission_id>
        <serial_number>Edited</serial_number>
        <state>Active</state>
        <unit>EA</unit>
        <part_number>I-013673-Parent</part_number>
      </Item>
    </Result>
    <Message>
      <event name="ids_modified" value="CFD2956084B7441285F10F070BD8A62F" />
    </Message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 36.

An error is raised by default if the `part` property value is a **`Part Number`** or if another foreign property is given in the `where` attribute clause of the `edit` request. The `where` clause is restricted to the **`ItemType`** given in the `type` attribute for security reasons. The `skipValidation` attribute does not work for the `where` attribute.

```
<AML>
  <Item type='PhysicalPart' action='edit' where="[PhysicalPart].part_number='I-013673-Parent'">
  <serial_number>Edited</serial_number>
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault xmlns:af="http://www.aras.com/InnovatorFault">
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring><![CDATA[Invalid column name 'part_number'.]]></faultstring>
      <detail>
        <af:legacy_detail><![CDATA[Invalid column name 'part_number'.]]></af:legacy_detail>
        <af:exception message="Invalid column name 'part_number'." type="System.Data.SqlClient.SqlException" />
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 37.

### 6.3.5 PhysicalPart Item Delete Requests with Foreign Properties

A `delete` request works if the `part` property is an **`ID`** of a related **Part** Item in the `where` attribute clause.

```
<AML>
  <Item type='PhysicalPart' action='delete' where="[PhysicalPart].part='A21521ED0D2648228757C77E9FE55625'">
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Result />
    <Message>
      <event name="ids_modified" value="737452E1AC9041B7865204532529A796|B5993D89B5814798A6CB4E36902175B2|CF
    </Message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 38.

An error is raised by default if the `part` property value is a **`Part Number`** or if another foreign property is given in the `where` attribute clause of the `delete` request. The `where` clause is restricted to the **`ItemType`** given in the `type` attribute for security reasons. The `skipValidation` attribute does not work for the `where` attribute.

```
<AML>
  <Item type='PhysicalPart' action='delete' where="[PhysicalPart].part_number='I-013673-Parent'">
  </Item>
</AML>
```

Tools Code Report Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault xmlns:af="http://www.aras.com/InnovatorFault">
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring><![CDATA[Invalid column name 'part_number'.]]></faultstring>
      <detail>
        <af:legacy_detail><![CDATA[Invalid column name 'part_number'.]]></af:legacy_detail>
        <af:exception message="Invalid column name 'part_number'." type="System.Data.SqlClient.SqlException" />
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 39.

## 6.4 Configuring inventory IDs of PhysicalPart Items

Before DTC 12.0R3, all **Serial Number** and **Lot / Batch** property values of the **Physical Part** Items in any State must be unique across a given **Part** Item. Additionally, a **Physical Part** Item in the **Active** State must have such a value.

However, knowing or identifying an entire inventory ID (serial or lot number) of a real-world asset is not always possible. For example, such a number can be lost entirely or partly due to wear, not assigned yet, not received yet, and so on.

Thus, the DTC 12.0R3 application is enhanced with settings that allow a serial-controlled or lot-controlled **Physical Part** Item in any State to have either a unique system-generated or duplicating **Serial Number** or **Lot / Batch** value.

The enhancement implementation is as follows:

- The **php_SerialLotUniquenessPolicy** global Variable for configuring the uniqueness of the **Serial Number** and **Lot / Batch** values across a given **Part Number** in an organization.
- Some server-side logic that governs this uniqueness according to the Variable setting.
- The **php_UnknownSerialNumber** Sequence for generating dummy **Serial Number** property values where real serial numbers are unknown.
- The **php_UnknownLotNumber** Sequence for generating dummy **Lot / Batch** property values where real lot numbers are unknown.

### 6.4.1 php_SerialLotUniquenessPolicy global Variable

The **php_SerialLotUniquenessPolicy** global Variable is a tool for turning on or off the uniqueness of the **Serial Number** and **Lot / Batch** property values of the **Physical Part** Items in any State across a given associated **Part** Item in an organization.

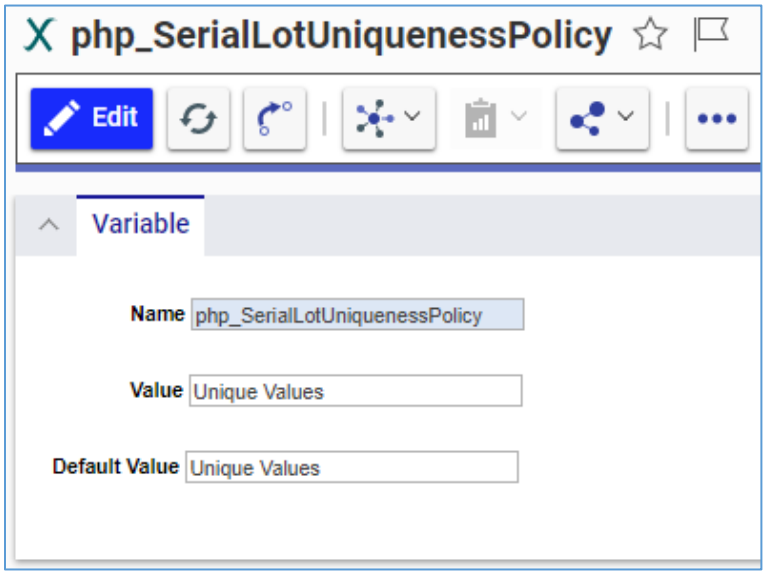



Figure 40.

With this tool, you can configure the uniqueness in question as follows:

- On—*only one* **Physical Part** Item in any State across *one given* associated **Part** Item in an organization can have *one given* **Serial Number** or **Lot / Batch** value.
- Off—*more than one* **Physical Part** Items in any State across *one given* associated **Part** Item in an organization can have *one given* **Serial Number** or **Lot / Batch** value.

**Warning** It is highly recommended not to change the **php_SerialLotUniquenessPolicy** Variable setting when a database contains **Physical Part** Items to avoid erroneous application behavior and data corruption.

When the uniqueness is turned on with the **php_SerialLotUniquenessPolicy** Variable, the server side forbids to save a serial- or lot-controlled **Physical Part** Item in any State with a **Serial Number** or **Lot / Batch** property value that has been already given to another serial- or lot-controlled **Physical Part** Item originated from the same **Part** Item. If trying to do so, the server raises an appropriate error message, either:

- A **Physical Part** Item with the same **Serial Number** for the same **Part** already exists.
- A **Physical Part** Item with the same **Lot Number** for the same **Part** already exists.

When the uniqueness is turned off, the server side allows to save a serial- or lot-controlled **Physical Part** Item in any State with a **Serial Number** or **Lot / Batch** property value that has been already given to another serial- or lot-controlled **Physical Part** Item originated from the same **Part** Item. The server does not raise an error message if trying to do so.

Out of the box, the Variable is configured as follows:

- **Value**: **Unique Values**
- **Default Value**: **Unique Values**

The **php_SerialLotUniquenessPolicy** Variable supports the following values:

- **Unique Values**
- **Nonunique Values**
- **Sequenced Values**

Use these values to configure the Variable.

The server side switches off or on the uniqueness per the Variable setting as follows:

1. The server-side logic checks whether the **Value** property has a supported value set. If it has, the server-side logic toggles the uniqueness according to this supported value as follows:
    - **Unique Values**: the uniqueness is on.
    - **Sequenced Values**: the uniqueness is on.
    - **Nonunique Values**: the uniqueness is off.
2. If the **Value** property is empty or set to an unsupported value, server-side logic checks a value set in the **Default Value** property and toggles the uniqueness accordingly:
    - **Unique Values**: the uniqueness is on.
    - **Sequenced Values**: the uniqueness is on.
    - **Nonunique Values**: the uniqueness is off.
    - Null (no value): the uniqueness is on.
    - An unsupported value: the uniqueness is on.

Any of the abovementioned settings do not affect the ability of *one or more serial-* or *lot-controlled* **Physical Part** Items within *one given* **Part Number** to have their **Serial Number** or **Lot / Batch** properties *empty* in the **Preliminary** State. In the **Active** State, a given **Serial Number** or **Lot / Batch** property must have a value.

Besides switching on the uniqueness, the **php_SerialLotUniquenessPolicy** Variable set to the **Sequenced Values** value also turns on the automatic population of the blank **Serial Number** and **Lot / Batch** properties with dummy values generated with the **php_UnknownSerialNumber** and **php_UnknownLotNumber** Sequences. In this configuration, when a user saves a **Physical Part** Item in any State with an empty **Serial Number** or **Lot / Batch** property, the system populates this empty property with a unique value generated from the **php_UnknowSerialNumber** or **php_UnknowLotNumber** Sequence if this **Physical Part** Item has its **unknown_number** property set to **true**.

The **unknown_number** property (the **Unknown** check box) does not affect the uniqueness and its toggling.

### 6.4.2 php_UnknownSerialNumber Sequence

The **php_UnknownSerialNumber** Sequence can be used for generating dummy values for the **Serial Number** properties of the serial-controlled **Physical Part** Items where real serial numbers of assets are not known.

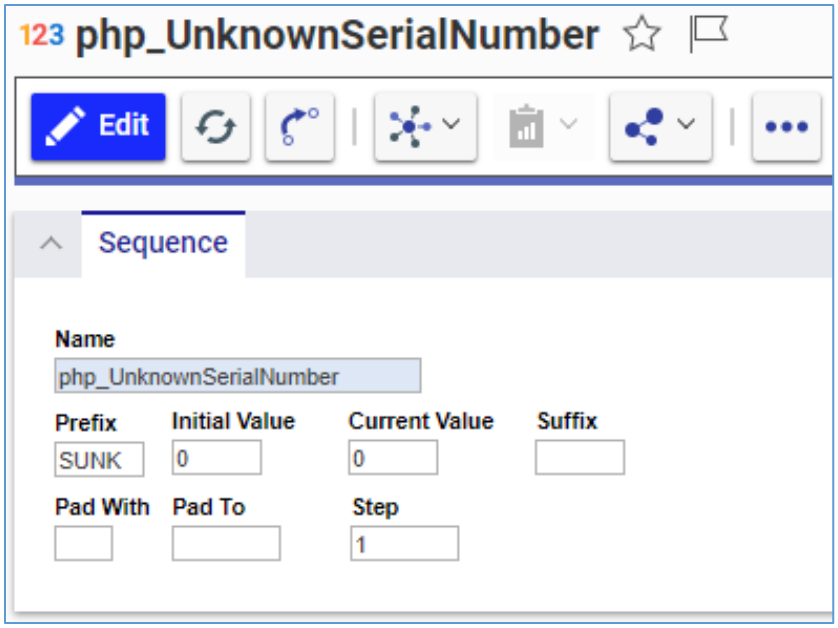



Figure 41.

The default configuration of the **php_UnknownSerialNumber** Sequence is the following:

- **Prefix**: **SUNK**
- **Initial Value**: **0**
- **Current Value**: **0**
- **Suffix**: Null (no value)
- **Pad With**: Null (no value)
- **Pad To**: Null (no value)
- **Step**: **1**

You can configure the default Sequence configuration as your organization requires.

To populate a **Serial Number** property automatically with a value generated by the system according to this Sequence, configure the **php_SerialLotUniquenessPolicy** global Variable to **Sequenced Values**.

Please keep in mind that this setting also switches on the **php_UnknownLotNumber** Sequence that generates dummy values for the **Lot / Batch** properties of the lot-controlled **Physical Part** Items where lot numbers are unknown. Currently, the **php_UnknownSerialNumber** Sequence cannot be switched on or off separately from the **php_UnknownLotNumber** Sequence.

The **php_UnknownSerialNumber** Sequence is global: the system outputs values using it for all serial-controlled **Physical Part** Items regardless of their associated **Part** Item. **Physical Part** Items rooted in different **Part** Items can have dummy **Serial Number** property values generated by the discussed Sequence.

### 6.4.3 php_UnknownLotNumber Sequence

The **php_UnknownLotNumber** Sequence can be used for generating dummy values for the **Lot / Batch** properties of the lot-controlled **Physical Part** Items where real lot numbers of assets are not known.

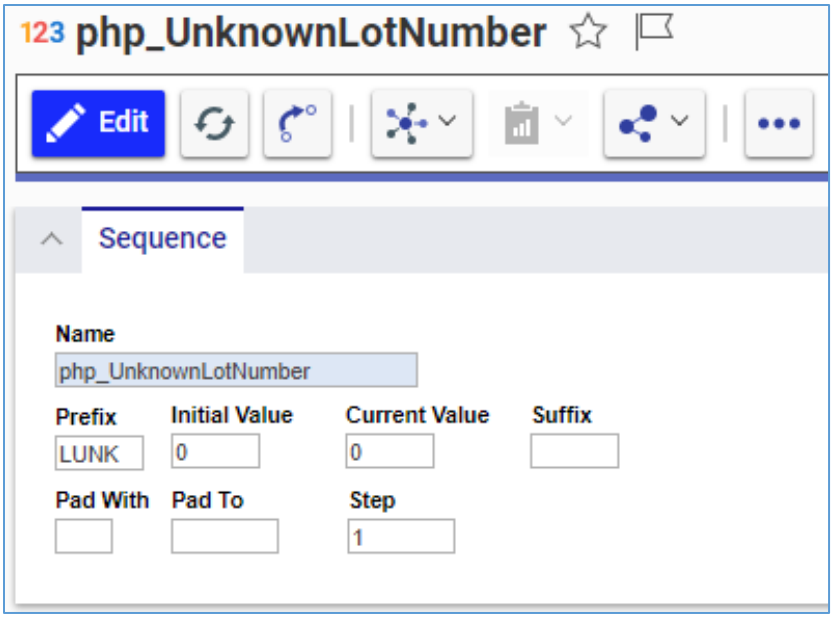



Figure 42.

The default configuration of the **php_UnknownLotNumber** Sequence is the following:

- **Prefix**: **LUNK**
- **Initial Value**: **0**
- **Current Value**: **0**
- **Suffix**: Null (no value)
- **Pad With**: Null (no value)
- **Pad To**: Null (no value)
- **Step**: **1**

You can configure the default Sequence configuration as your organization requires.

To populate a **Lot / Batch** property automatically with a value generated by the system according to this Sequence, configure the **php_SerialLotUniquenessPolicy** global Variable to **Sequenced Values**. Please keep in mind that this setting also switches on the **php_UnknownSerialNumber** Sequence that generates dummy values for the **Serial Number** properties of the serial-controlled **Physical Part** Items where serial numbers are unknown. Currently, the **php_UnknownLotNumber** Sequence cannot be switched on or off separately from the **php_UnknownSerialNumber** Sequence.

The **php_UnknownLotNumber** Sequence is global: the system outputs values using it for all lot-controlled **Physical Part** Items regardless of their associated **Part** Item. **Physical Part** Items rooted in different **Part** Items can have dummy **Lot / Batch** property values generated by the discussed Sequence.

# 7 Physical Part BOM Items

## 7.1 Physical Part BOM Item Overview

A **Physical Part BOM** Item is a Relationship of the **PhysicalPart BOM** RelationshipType that represents a Parent-Child (assembly-component) relationship between two **Physical Part** Items.

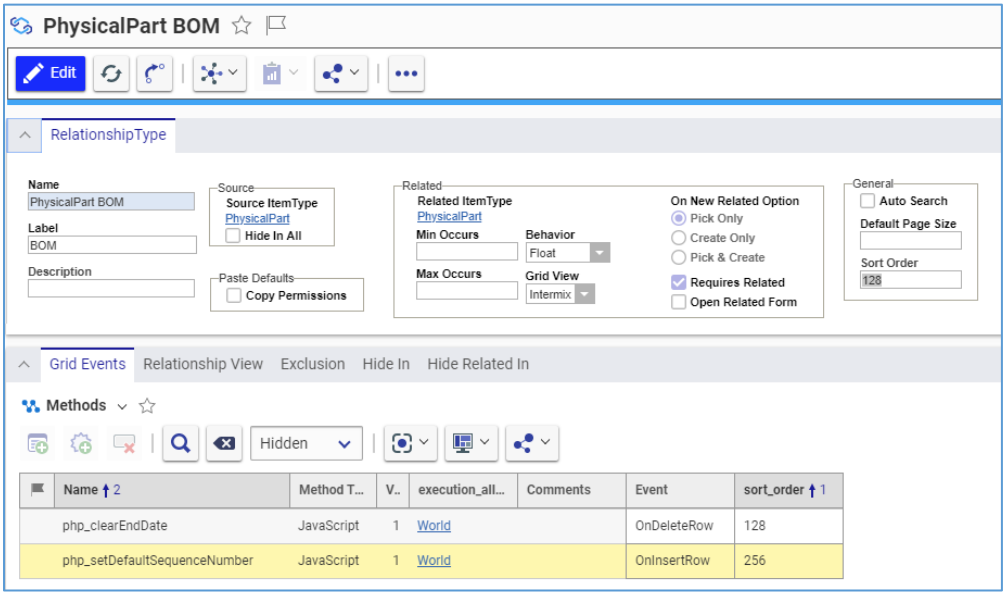



Figure 43.

The **PhysicalPart BOM** RelationshipType has a corresponding **PhysicalPart BOM** ItemType.

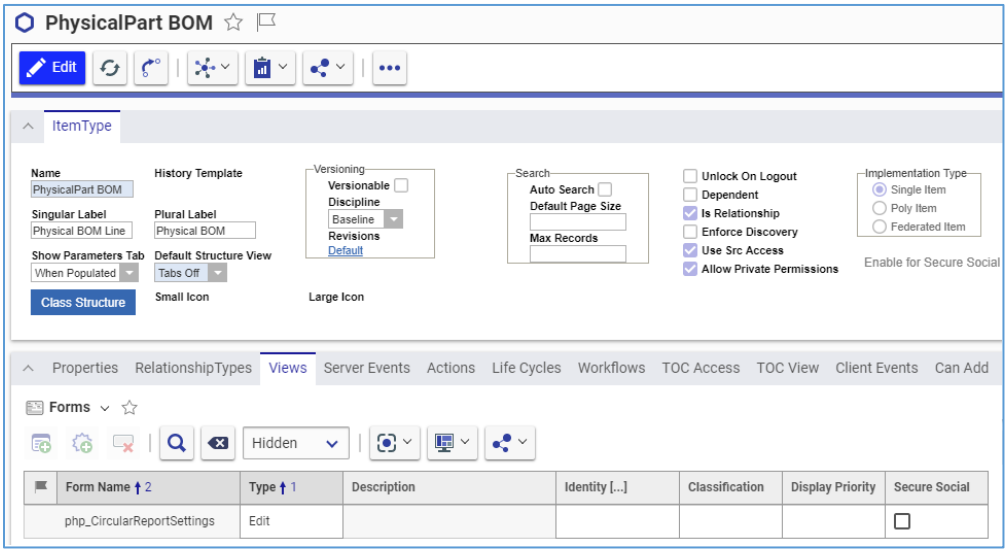



Figure 44.

The **PhysicalPart BOM** properties should be accessible only via **Physical Part BOM** Relationship Grids.

The **PhysicalPart BOM** Item is not versionable and has no Item view. Enabling versioning for the **PhysicalPart BOM** ItemType must be avoided as it may decrease application performance, including the output of large amounts of data.

The system keeps records of user Identities who created and were last to modify a **Physical Part BOM** Item. It shows them in the **Physical Part BOM** Relationship Grid.

## 7.2 Remove-and-Replace and AML

The Remove-and-Replace (R&R) operation is designed and implemented as a piece of GUI functionality for dedicated end-users: members of the **Asset User**, **Asset Editor**, and **Asset Admin** Identities. It is a digital representation of a real-world assembly component replacement that keeps records on replaced and replacing components along with identifying their belonging to the same installation in the assembly.

It is highly recommended to perform R&R operations with GUI. However, if your organization has a strong need to perform them with AML, you should tackle how the client side generates R&R AML requests.

The *Aras Digital Twin Core 12.0R3 – User Guide* describes all details, requirements, constraints, and valid data values.

In addition to the required properties, an R&R AML request should have the following virtual properties:

- `is_remove_replace`: a required Boolean property of a parent (source) **PhysicalPart** Item that indicates an R&R operation if set to `1`.
- `replaced_id`: a required **PhysicalPart BOM** Relationship Item property that keeps a list of GUIDs of **PhysicalPart BOM** Relationship Items that will be replaced.
- `replacing_id`: a required **PhysicalPart BOM** Relationship Item property that keeps a list of GUIDs of **PhysicalPart BOM** Relationship Items will replace Items.
- `is_split_rest`: an indicator of a new **Physical Part BOM** Relationship Item with the *remaining* **Quantity** property value. This virtual property is required only in the case of partial splitting of the *original* **Quantity** property value. Only one in a request can have this property.

The *19.1 Remove-and-Replace AML Replication Example* section provides some AML examples of various R&R cases. These examples should give you a general understanding of composing R&R AML requests and may not work if you just run them.

# 8 Life Unit Items

## 8.1 Life Unit Permissions

The default Permission for the **Life Unit** Items is **php_LifeUnit**.

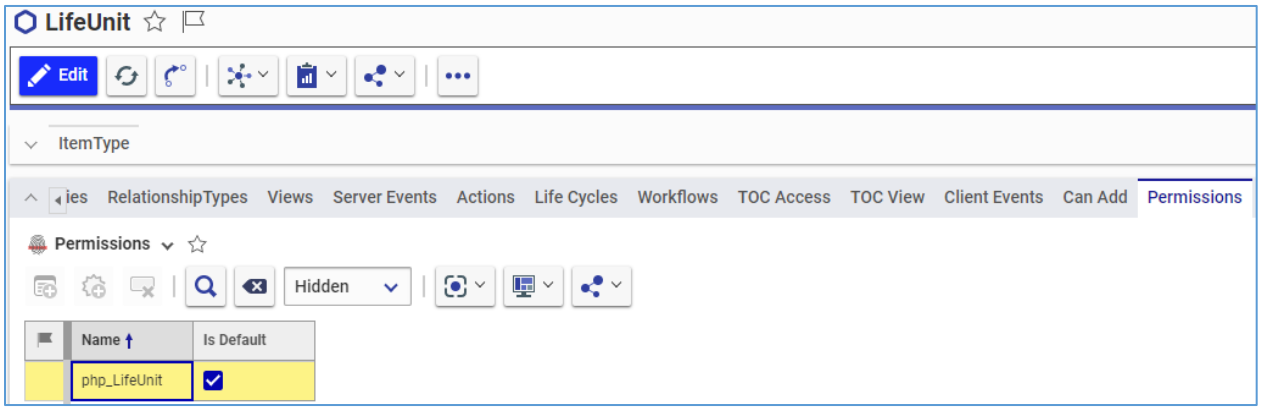



Figure 45.

It grants full Access Rights only to the **Asset Admin** Identity. All other Asset Identities have only **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights.

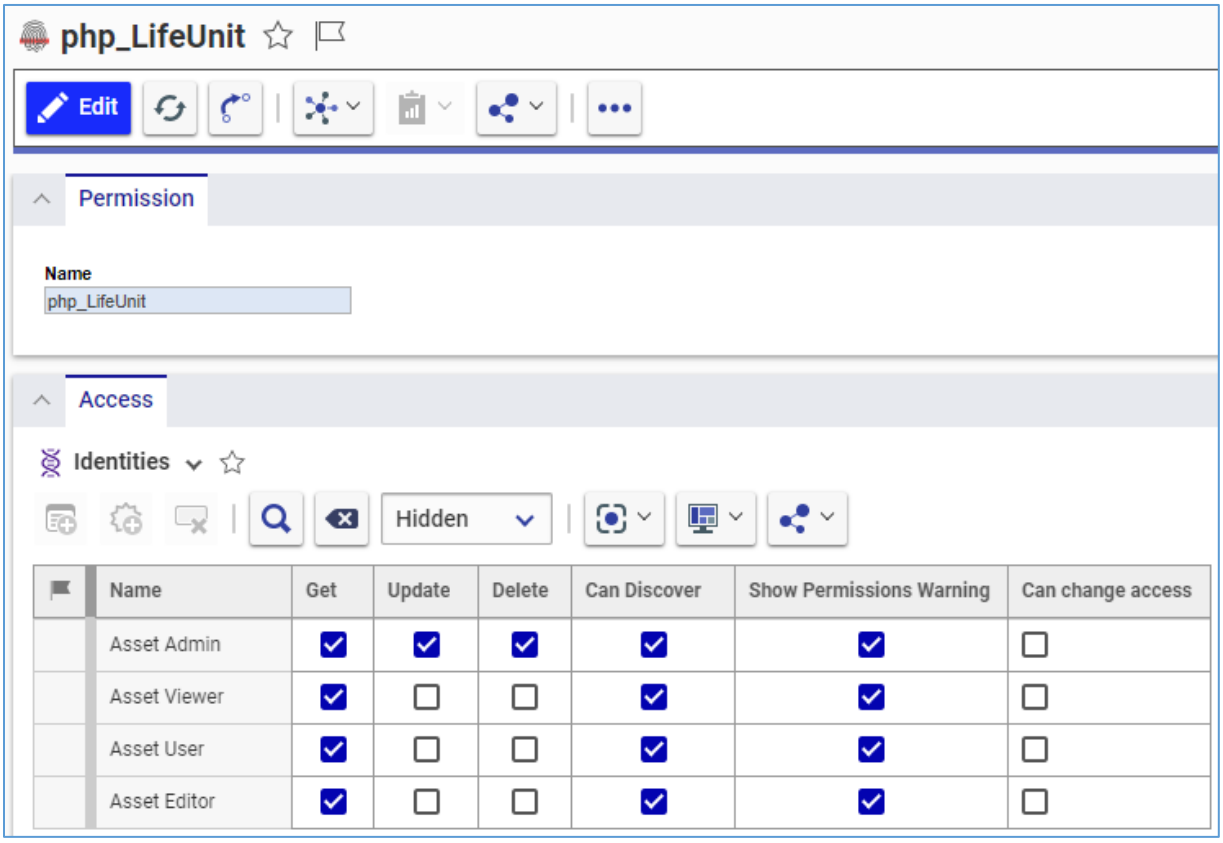



Figure 46.

The **Life Unit** ItemType **Can Add** is set to the **Asset Admin** Identity.

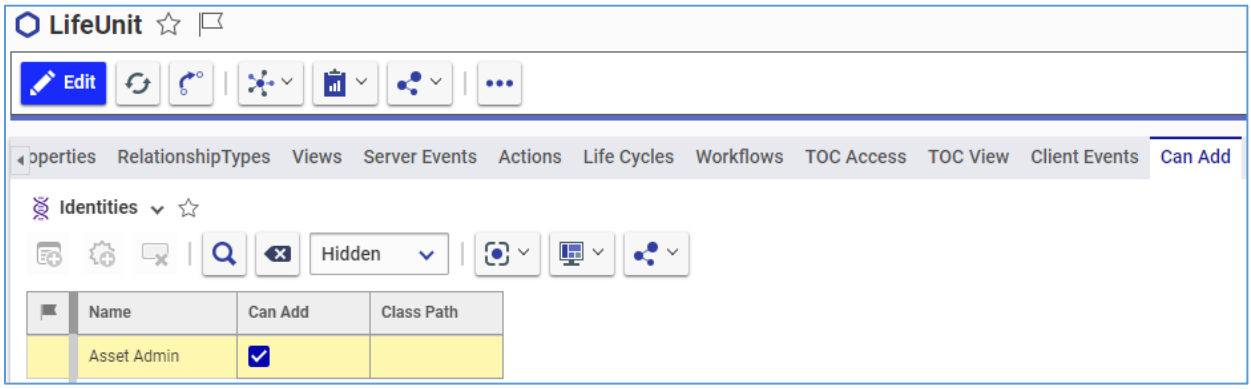



Figure 47.

# 9 Life Parameter Items

A **Life Parameter** Item should represent a life characteristic (variable) that is generic to as many organization assets as possible. An organization is supposed to have a reasonably small number of the **Life Parameter** Items.

Being a highly reused Item, a **Life Parameter** Item is supposed to be created, updated, and deleted individually by an **Asset Admin** via Aras Innovator UI with a deep understanding of the Item impact.

## 9.1 Life Parameter Permissions

The default Permission for the **Life Parameter** Items is **php_PreliminaryLifeParameter**.

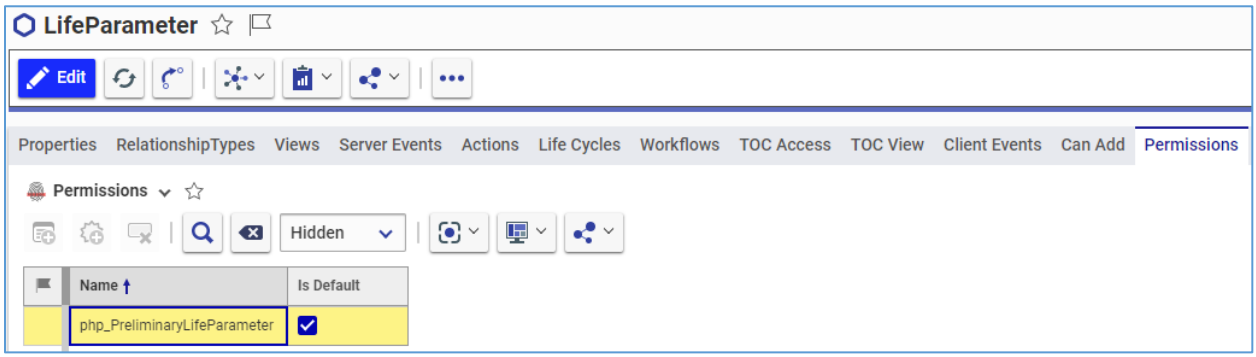



Figure 48.

The **LifeParameter** Life Cycle Map sets Permissions for the **Life Parameter** Items depending on their States as described in the following table. The **Promoted by** column shows the Identities that can promote an Item to a given Life Cycle State. The **Permission** column refers to a Permission that manages Access Rights in a given Life Cycle State.

Table 2: The Life Parameter Permissions matrix

| Life Cycle State | Can Add | Update | Delete | Change access | Promoted by | Permission |
|---|---|---|---|---|---|---|
| Preliminary | Asset Admin | Asset Admin | Asset Admin | No one | Asset Admin | php_PreliminaryLifeParameter |
| Active | NA | No one | No one | No one | Asset Admin | php_ActiveLifeParameter |

**Note:** All the Asset Identities have **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights for the **Life Parameter** Items in any Item State.

## 9.2 Life Parameters and AML

**Warning**: Creating or updating the **Life Parameter** Items via AML must be avoided because only the client side can validate the behavior of their properties. For the behavior details, refer to the dedicated section in the *Aras Digital Twin Core 12.0R2 – User Guide*.

If it is necessary to create or update **Life Parameter** Items via AML, you must manually check each property of each Item to get a valid value.

```
<AML>
  <Item type='LifeParameter' action='add'>
    <control_type>Serial</control_type>
    <hard_life_indicator>1</hard_life_indicator>
    <name>Generator Running Hours</name>
    <reset_trigger>Yes</reset_trigger>
    <scale>0</scale>
    <shelf_life_control>Recert</shelf_life_control>
    <unit>Hours</unit>
    <item_number>GRHRS</item_number>
  </Item>
</AML>
```

Tools | Code | Report | Table

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Result>
      <Item type="LifeParameter" typeId="228216EFFE2A47DA9161C678397CC7A1" id="E19B57DF1FF5402EBCE0B93FA8296BF4">
        <config_id keyed_name="GRHRS" type="LifeParameter">E19B57DF1FF5402EBCE0B93FA8296BF4</config_id>
        <control_type>Serial</control_type>
        <created_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</created_by_id>
        <created_on>2020-11-18T13:52:23</created_on>
        <current_state name="Preliminary" keyed_name="Preliminary" type="Life Cycle State">4249677785F94A83B2925FEFC9DF0DAF</current_state>
        <generation>1</generation>
        <hard_life_indicator>1</hard_life_indicator>
        <id keyed_name="GRHRS" type="LifeParameter">E19B57DF1FF5402EBCE0B93FA8296BF4</id>
        <is_current>1</is_current>
        <is_released>0</is_released>
        <keyed_name>GRHRS</keyed_name>
        <major_rev>A</major_rev>
        <modified_by_id keyed_name="Innovator Admin" type="User">30B991F927274FA3829655F50C99472E</modified_by_id>
        <modified_on>2020-11-18T13:52:23</modified_on>
        <name>Generator Running Hours</name>
        <new_version>1</new_version>
        <not_lockable>0</not_lockable>
        <permission_id keyed_name="php_PreliminaryLifeParameter" type="Permission">E515EB0ED3AE4524BC4548F400C9707B</permission_id>
        <reset_trigger>Yes</reset_trigger>
        <scale>0</scale>
        <shelf_life_control>Recert</shelf_life_control>
        <state>Preliminary</state>
        <unit>Hours</unit>
        <item_number>GRHRS</item_number>
      </Item>
    </Result>
    <Message>
      <event name="ids_modified" value="E19B57DF1FF5402EBCE0B93FA8296BF4" />
    </Message>
  </SOAP-ENV:Body>
```

Figure 49.

The **Life Parameter** Item view fields that contain invalid values are blank.

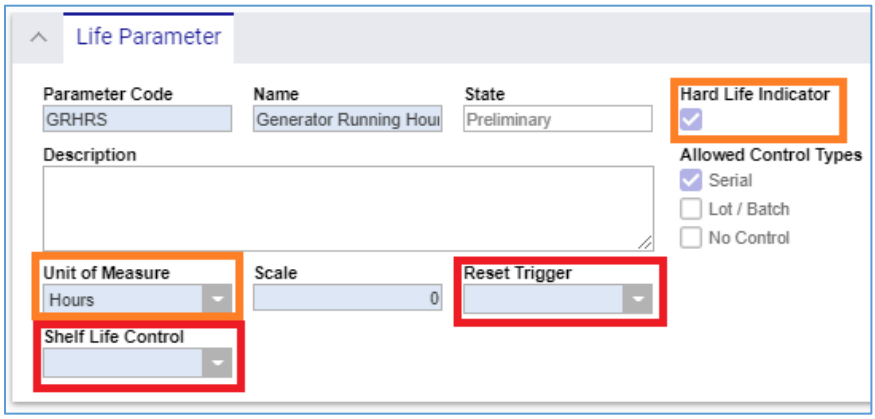



Figure 50.

It is possible to promote a **Life Parameter** Item with invalid values to the **Active** State.

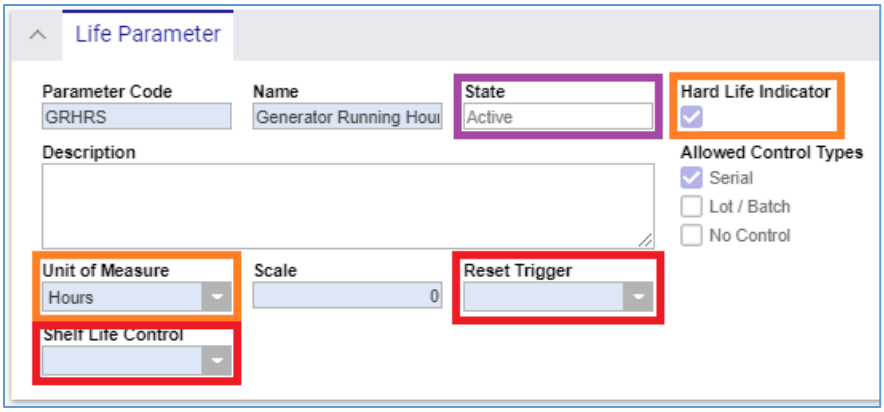



Figure 51.

If such an Item is spotted, correct it by editing it in the **Preliminary** state. Once the Item enters the in-editing mode, the client side will automatically correct the given values according to the built-in business logic. Continue editing the values if the automatic correction is not enough.

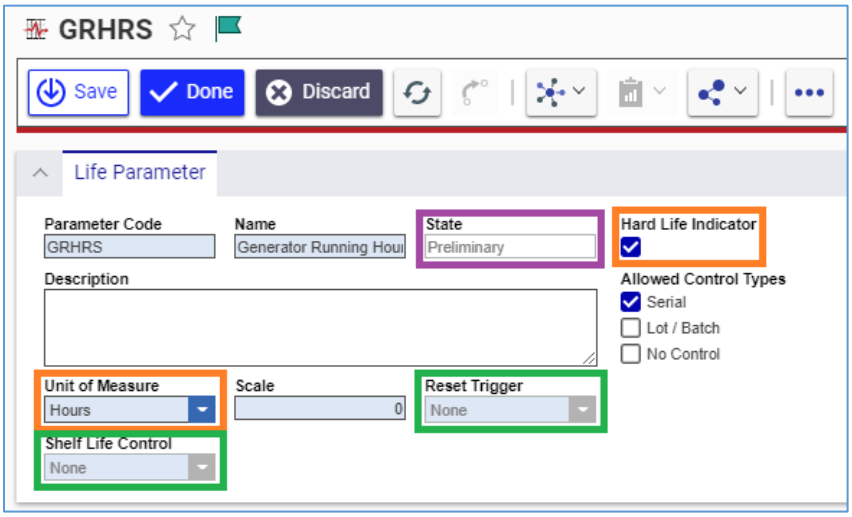



Figure 52.

# 10 Life Policy Items

A **Life Policy** Item should represent a set of life characteristics generic to as many organization assets as possible. An organization is supposed to have a reasonably small number of **Life Policy** Items.

Because this Item is frequently used, a **Life Policy** Item is supposed to be created, updated, and deleted individually by an **Asset Admin**.

## 10.1 Life Policy Permissions

The default Permission for the **Life Policy** Items is **php_PreliminaryLifePolicy**.

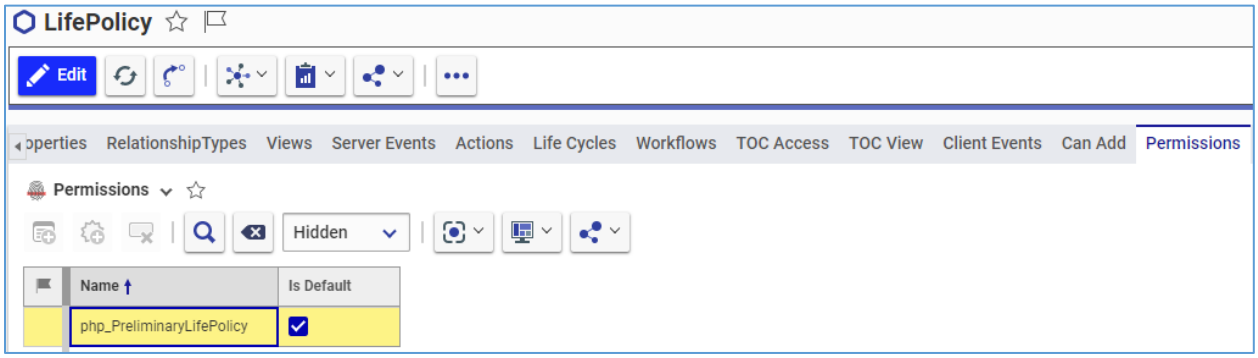



Figure 53.

The **LifePolicy** Life Cycle Map sets Permissions for the **Life Policy** Items depending on their States as described in the following table. The **Promoted by** column shows the Identities that can promote an Item to a given Life Cycle State. The **Permission** column refers to a Permission that manages Access Rights in a given Life Cycle State.

Table 3: The Life Parameter Permissions matrix

| Life Cycle State | Can Add | Update | Delete | Change access | Promoted by | Permission |
|---|---|---|---|---|---|---|
| Preliminary | Asset Admin | Asset Admin | Asset Admin | No one | Asset Admin | php_PreliminaryLifePolicy |
| Active | NA | No one | No one | No one | Asset Admin | php_ActiveLifePolicy |

**Note:** All the Asset Identities have the **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights for the **Life Policy** Items in any Item State.

# 11 Part Policy Items

## 11.1 Part Policy Permissions

The default Permission for the **Part Policy** Items is **php_PreliminaryPartPolicy**.

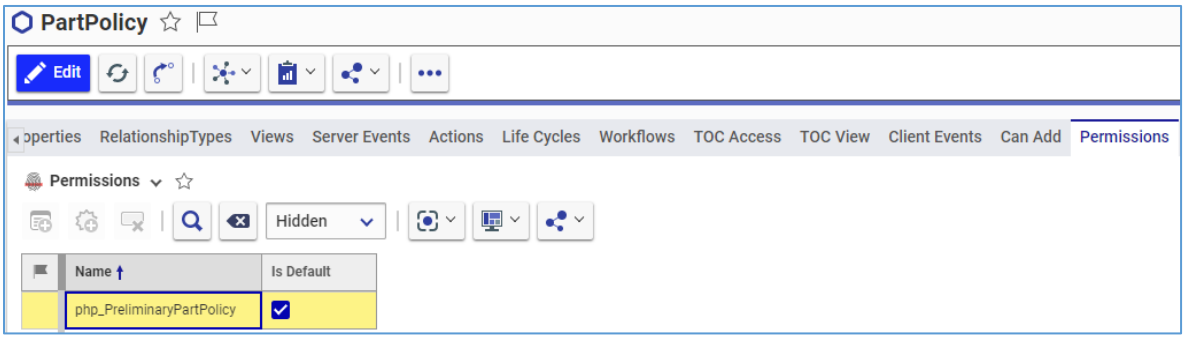



Figure 54.

The **PartPolicy** Life Cycle Map sets Permissions for the **Part Policy** Items depending on their States as described in the following table. The **Promoted by** column shows the Identities that can promote an Item to a given Life Cycle State. The **Permission** column refers to a Permission that manages Access Rights in a given Life Cycle State.

Table 4: The Life Parameter Permissions matrix

| Life Cycle State | Can Add | Update | Delete | Change access | Promoted by | Permission |
|---|---|---|---|---|---|---|
| Preliminary | Asset Editor, Asset Admin | Asset Editor, Asset Admin | Asset Editor, Asset Admin | No one | Asset Editor, Asset Admin | php_PreliminaryPartPolicy |
| Active | NA | No one | No one | No one | Asset Editor, Asset Admin | php_ActiveLifeParameter |

**Note:** All the Asset Identities, including **Asset Viewer**, have the **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights for the **Part Policy** Items in any Item State.

## 11.2 PartPolicy and AML

When creating a **Part Policy** Item via AML, you must set up the `control_type` property explicitly because it is not copied from a **Part** Item given in `part_id`.

```
<AML>
  <Item type='PartPolicy' action='add' doGetItem='0'>
    <part_id type="Part">PartID</part_id>
```

```
    <life_policy_id type="LifePolicy">LifePolicyID</life_policy_id>
    <control_type>Serial</control_type>
  </Item>
</AML>
```


Figure 55.

When you try to create a **Part Policy** Item using AML without the `control_type` property, an error is raised, and this **Part Policy** Item is not created.

```
<AML>
  <Item type='PartPolicy' action='add' doGetItem='0'>
    <part_id type="Part">PartID</part_id>
    <life_policy_id type="LifePolicy">LifePolicyID</life_policy_id>
  </Item>
</AML>
```


Figure 56.

# 12 PartPolicy LifeParameter Items

Being an automatically managed Relationship Item, a **PartPolicy LifeParameter** Item is supposed to be created and deleted individually by the **Aras PLM** Identity that represents the system. A user must not create or delete the **PartPolicy LifeParameter** Item. Doing so will break the DTC application logic of the **Physical Part** Items having only the **Life Parameter** Items that are in a **Life Policy** Item related to a **Part Policy** Item from which these **Life Parameter** Items originate.

## 12.1 PartPolicy LifeParameter Permissions

The default Permission for the **PartPolicy LifeParameter** Items is **php_PartPolicy LifeParameter**.

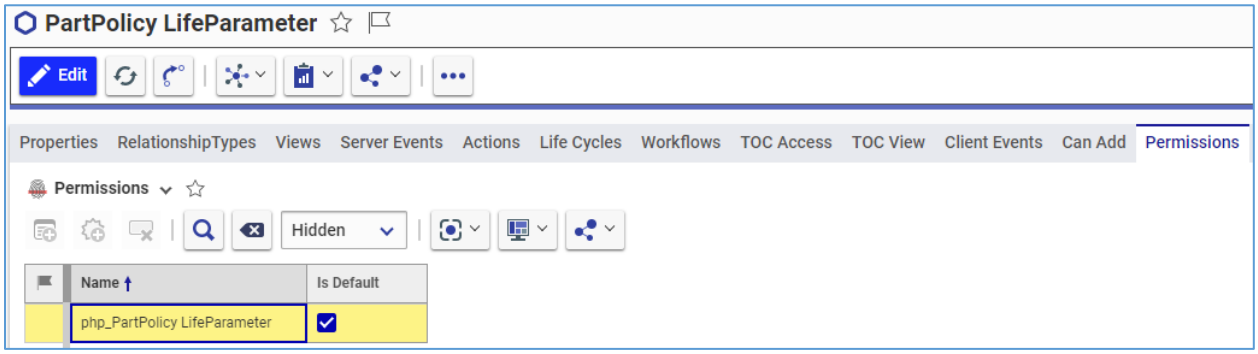



Figure 57.

It grants full Access Rights only to the **Aras PLM** Identity. All Asset Identities have **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights. The **Asset Admin** and **Asset Editor** Identities also have the **Update** Permission.

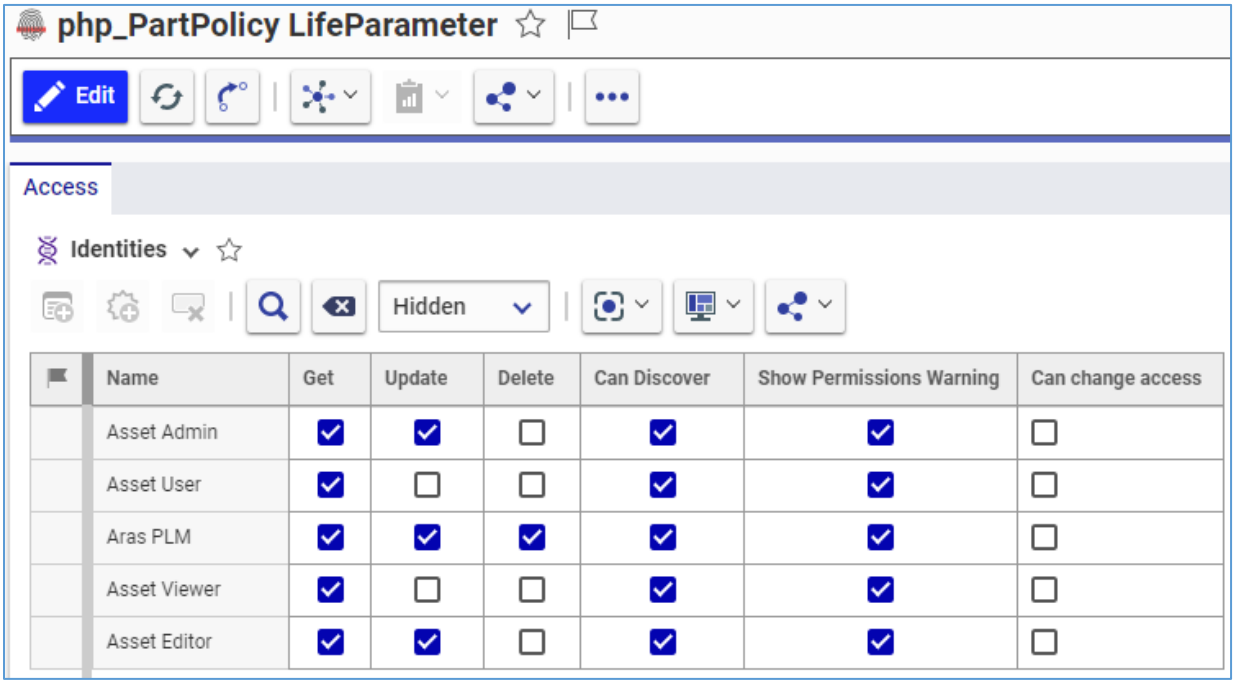



Figure 58.

The **PartPolicy LifeParameter** ItemType **Can Add** is set to the **Aras PLM** Identity.

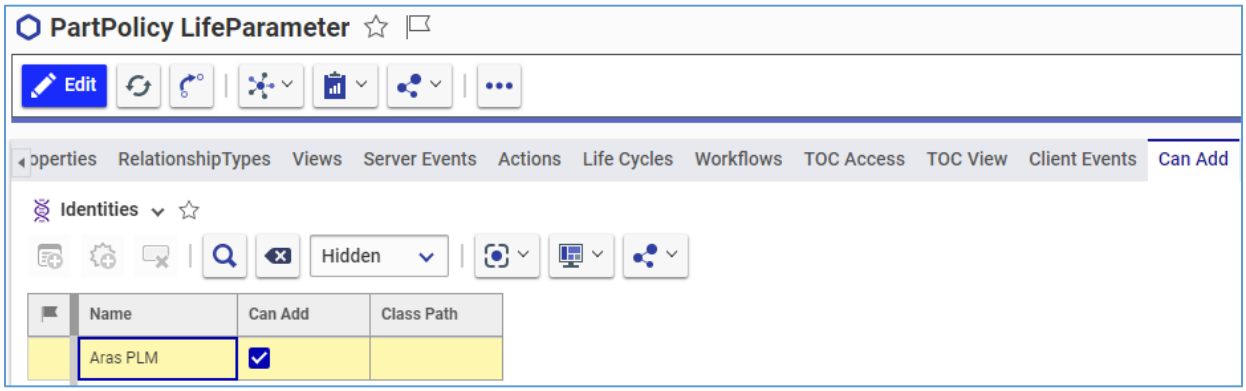



Figure 59.

## 12.2 PartPolicy LifeParameters and AML

To correctly handle and combine the Permissions of both **PartPolicy** and **PartPolicy LifeParameter** ItemTypes, a **PartPolicy LifeParameter** Item can be modified using AML only within the sourced **PartPolicy** Item context scope.

```
   <!--Modification in the Part Policy Context -->
<AML>
  <Item type='PartPolicy' action='edit' id='ID'>
    <Relationships>
      <Item type='PartPolicy LifeParameter' action='edit' id='ID>
      …
      </Item>
    </Relationships>
  </Item>
</AML>
```


Figure 60.

Direct editing of the **PartPolicy LifeParameter** Items using AML is not allowed because it can result in corrupt data.

```
   <!—Direct Modification-->
<AML>
  <Item type='PartPolicy LifeParameter' action='edit' id='ID>
     …
  </Item>
</AML>
```

When trying to edit a **PartPolicy LifeParameter** Item directly using AML, an error is raised, and this **PartPolicy LifeParameter** Item cannot be edited.

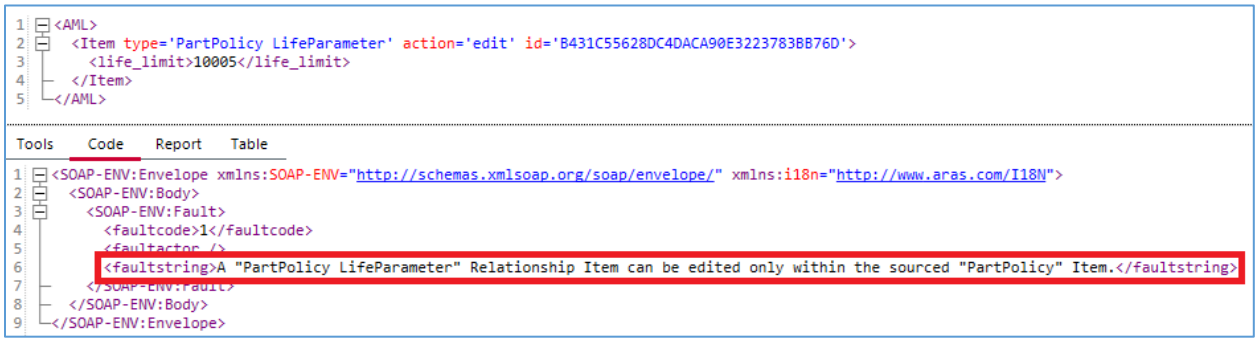



Figure 61.

# 13 PhysicalPart LifeValue Items

Because it is an automatically managed Relationship Item, a **PhysicalPart LifeValue** Item is supposed to be created and deleted individually by the **Aras PLM** Identity that represents the system. A user must not create or delete the **PhysicalPart LifeValue** Item because this will break the DTC application logic of the **Physical Part** Items having only the **Life Parameter** Items that are in a **Life Policy** Item related to a **Part Policy** Item from which these **Life Parameter** Items originate.

## 13.1 PhysicalPart LifeValue Permissions

The default Permission for the **PhysicalPart LifeValue** Items is **php_PhysicalPart LifeValue**.

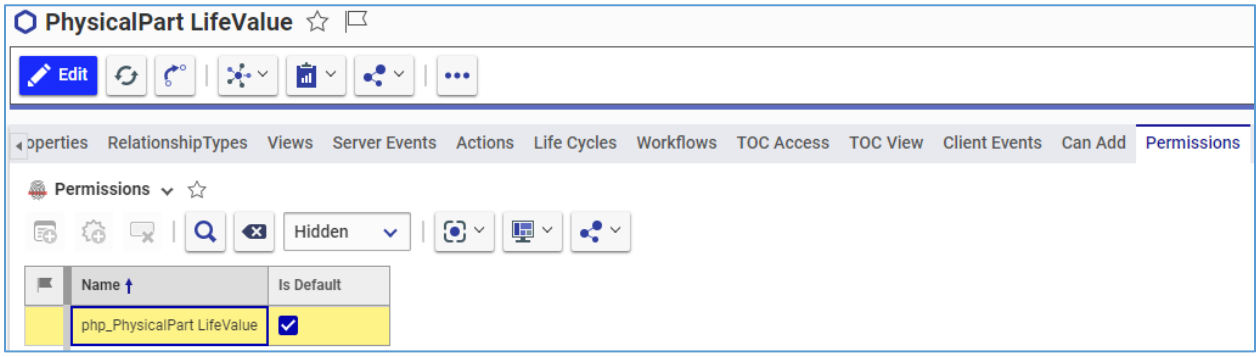



Figure 62.

It grants full Access Rights only to the **Aras PLM** Identity. All Asset Identities have the **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights. The **Asset User**, **Asset Admin**, and **Asset Editor** Identities also have **Update** Permission.

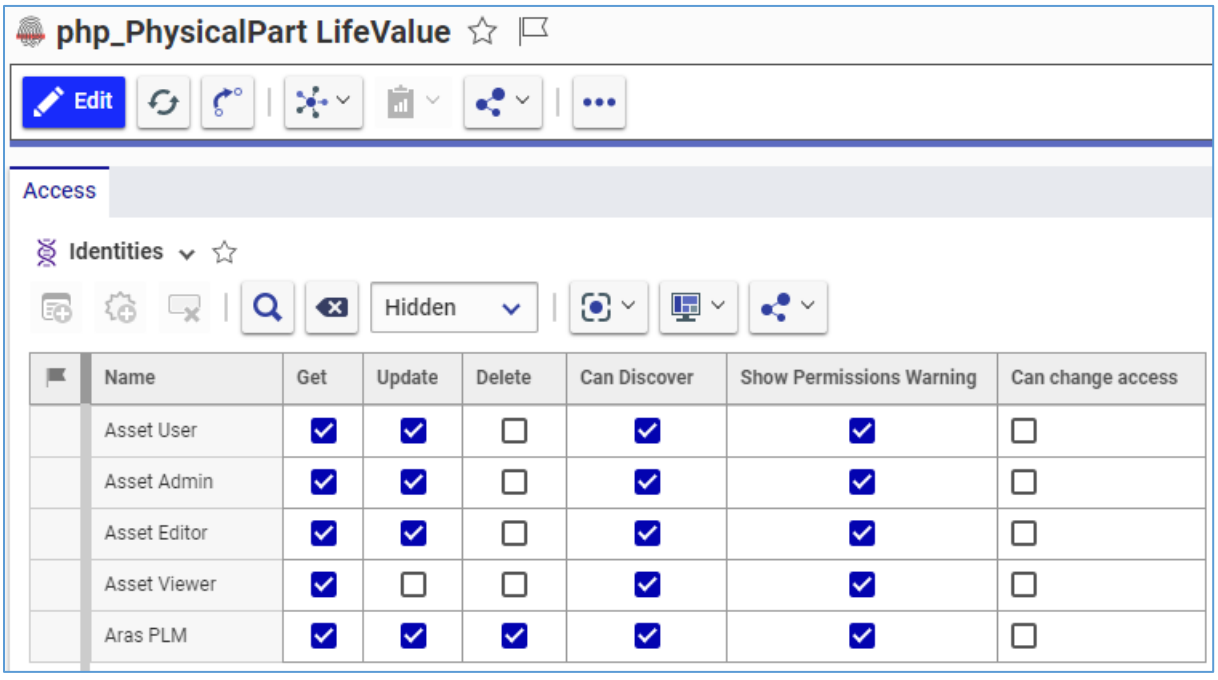



Figure 63.

The **PhysicalPart LifeValue** ItemType **Can Add** is set to the **Aras PLM** Identity.

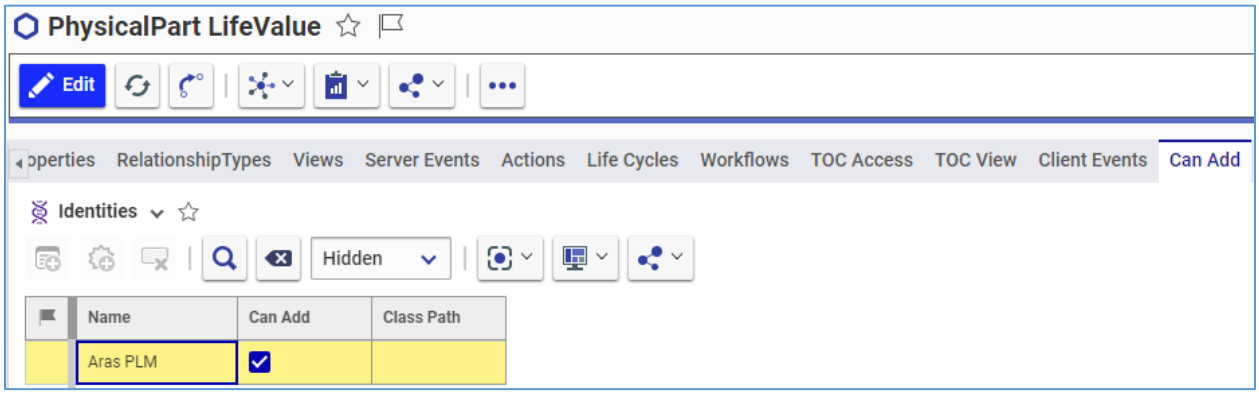



Figure 64.

## 13.2 PhysicalPart LifeValue and AML

To correctly handle and combine the Permissions of both **PhysicalPart** and **PhysicalPart LifeValue** ItemTypes, a **PhysicalPart LifeValue** Item should be modified using AML only within the sourced **PhysicalPart** Item context scope.

```
   <!--Modification in the PhysicalPart context -->
<AML>
  <Item type='PhysicalPart' action='edit' id='ID'>
    <Relationships>
      <Item type='PhysicalPart LifeValue' action='edit' id='ID>
      …
      </Item>
    </Relationships>
  </Item>
</AML>
```


Figure 65.

Direct editing of the **PhysicalPart LifeValue** Items using AML is possible but not recommended because it can corrupt data.

```
   <!—Direct Modification-->
<AML>
  <Item type='PhysicalPart LifeValue' action='edit' id='ID>
     …
  </Item>
</AML>
```


Figure 66.

# 14 PhysicalPart DateValue Items

Because it is an automatically managed Relationship Item, a **PhysicalPart DateValue** Item is supposed to be created and deleted individually by the **Aras PLM** Identity that represents the system. A user must not create or delete the **PhysicalPart LifeValue** Item because this will break the DTC application logic of the **Physical Part** Items having only the **Life Parameter** Items that are in a **Life Policy** Item related to a **Part Policy** Item from which these **Life Parameter** Items originate.

## 14.1 PhysicalPart DateValue Permissions

The default Permission for the **PartPolicy DateValue** Items is **php_PhysicalPart DateValue**.

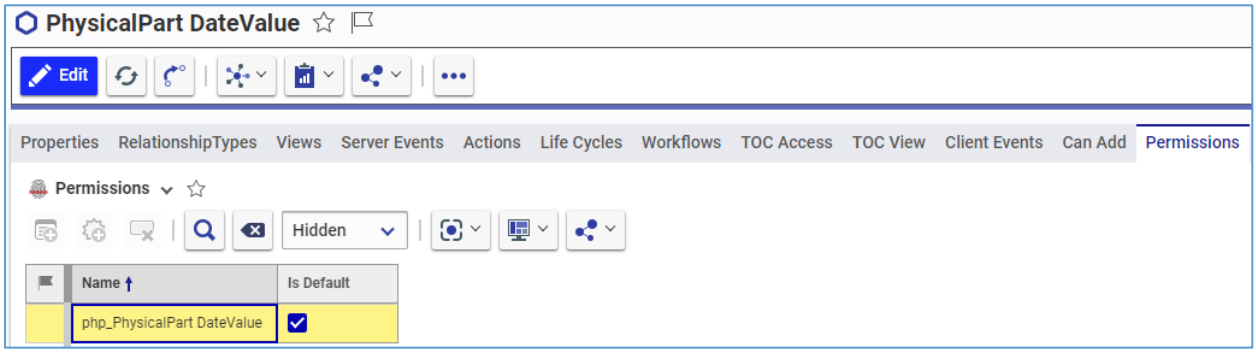



Figure 67.

It grants full Access Rights only to the **Aras PLM** Identity. All Asset Identities have the **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights. The **Asset User**, **Asset Admin**, and **Asset Editor** Identities also have **Update** Permission.

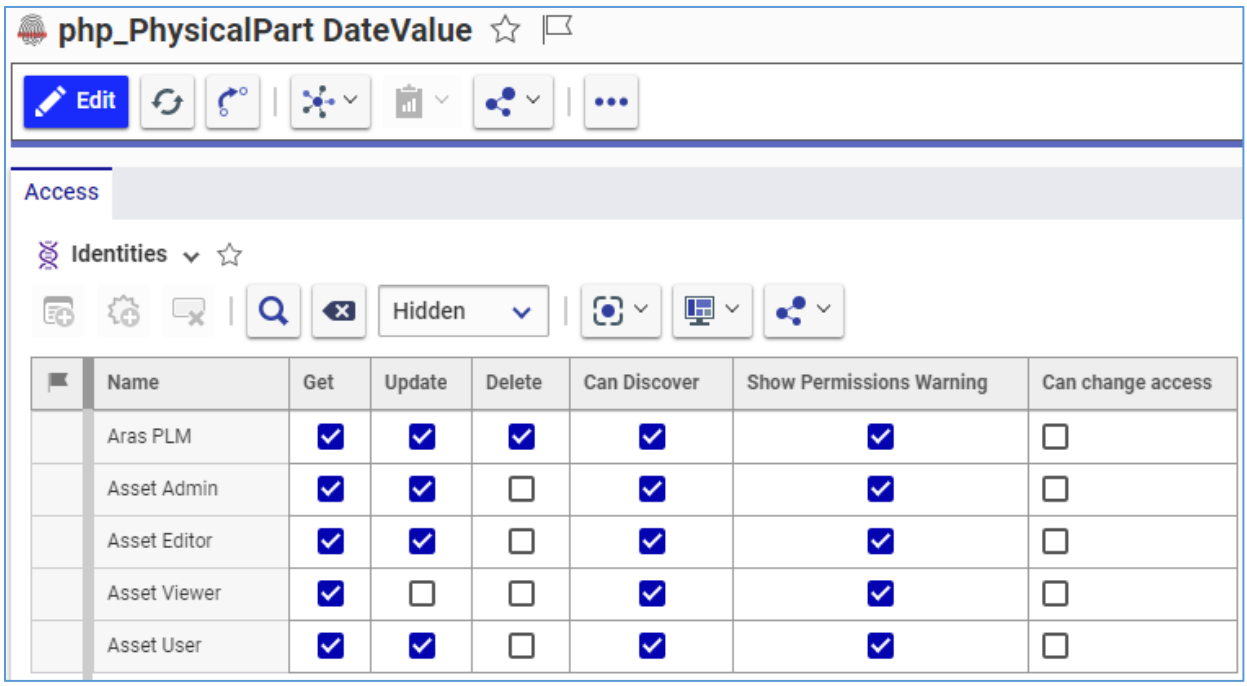



Figure 68.

The **PartPolicy DateValue** ItemType **Can Add** is set to the **Aras PLM** Identity.

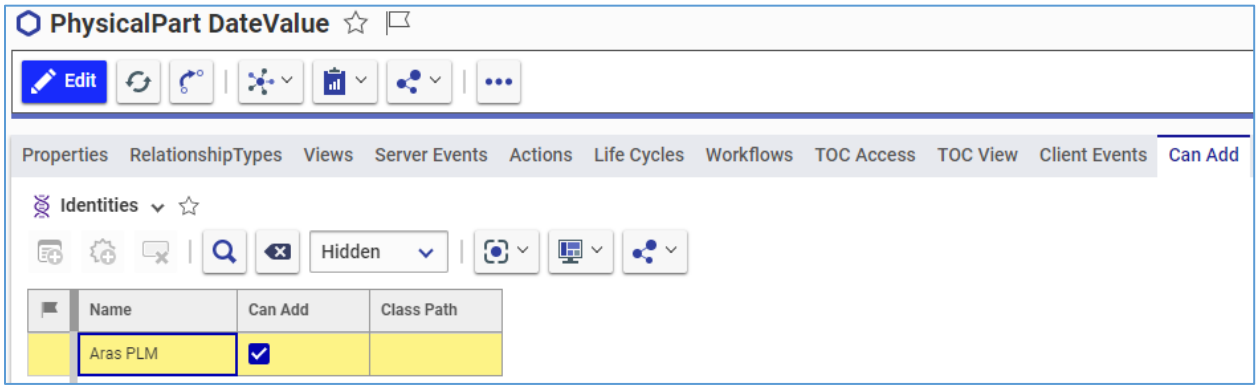



Figure 69.

## 14.2 PhysicalPart DateValue and AML

To correctly handle and combine the Permissions of both the **PhysicalPart** and **PhysicalPart DateValue** ItemTypes, a **PhysicalPart DateValue** Item should be modified using AML only within the sourced **PhysicalPart** Item context scope.

```
   <!--Modification in the PhysicalPart context -->
<AML>
  <Item type='PhysicalPart' action='edit' id='ID'>
    <Relationships>
      <Item type='PhysicalPart DateValue' action='edit' id='ID>
      …
      </Item>
    </Relationships>
  </Item>
</AML>
```


Figure 70.

Direct editing of the **PhysicalPart DateValue** Items using AML is possible but not recommended because data could be corrupted.

```
   <!—Direct Modification-->
<AML>
  <Item type='PhysicalPart DateValue' action='edit' id='ID>
     …
  </Item>
</AML>
```


Figure 71.

# 15 PhysicalPart LifeHistoryLog Items

As a **PhysicalPart LifeHistoryLog** Relationship Item represents a historical record of an asset life variable value, it should not be updated or deleted once created. Otherwise, the history of asset life will be corrupted, which may be critical in the industries where such data is rigorously controlled, like aerospace or the military.

Out of the box, the **PhysicalPart LifeHistoryLog** Items are fully managed automatically with UI or AML: only the **Aras PLM** Identity that represents the system can create, update, and delete such Items. A user must be able only to discover and view them.

## 15.1 PhysicalPart LifeHistoryLog Permissions

The default and only Permission for the **PhysicalPart LifeHistoryLog** Items is **php_PhysicalPart LifeHistoryLog**.

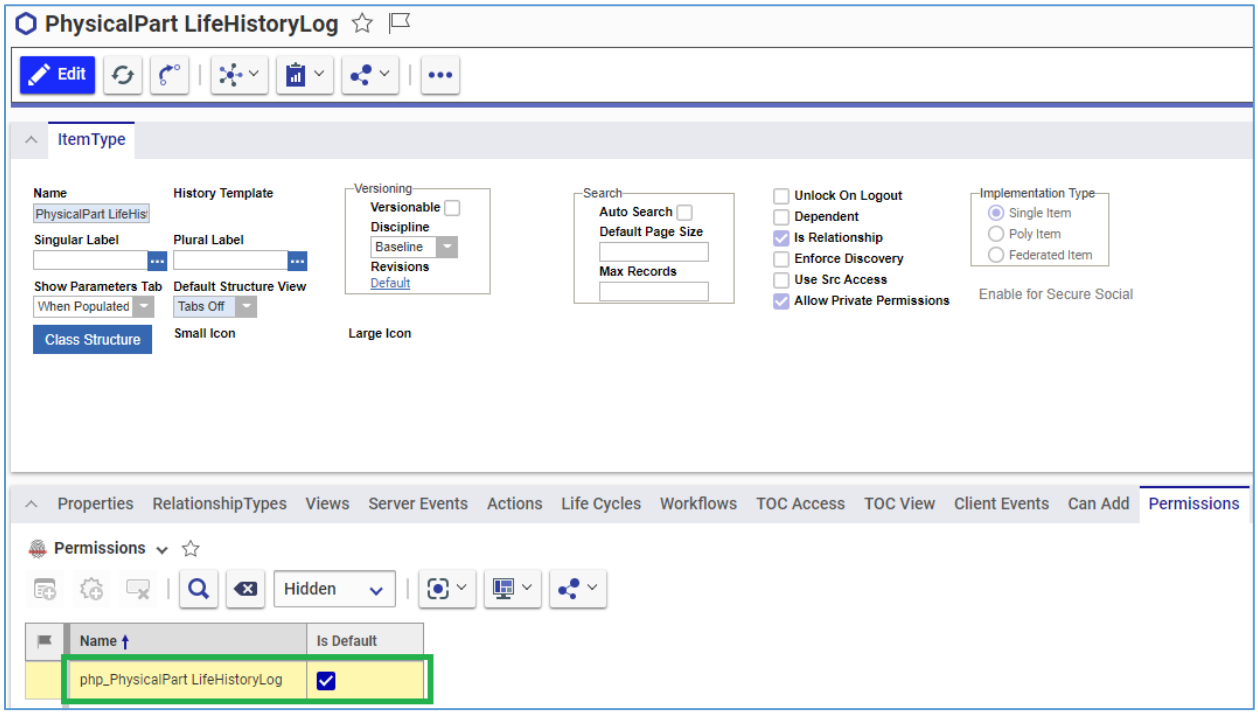



Figure 72.

It grants full Access Rights only to the **Aras PLM** Identity. The **Asset Viewer** Identity has the **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights. When granting Access Rights to other user Identities, ensure that they can only view the Items in question.

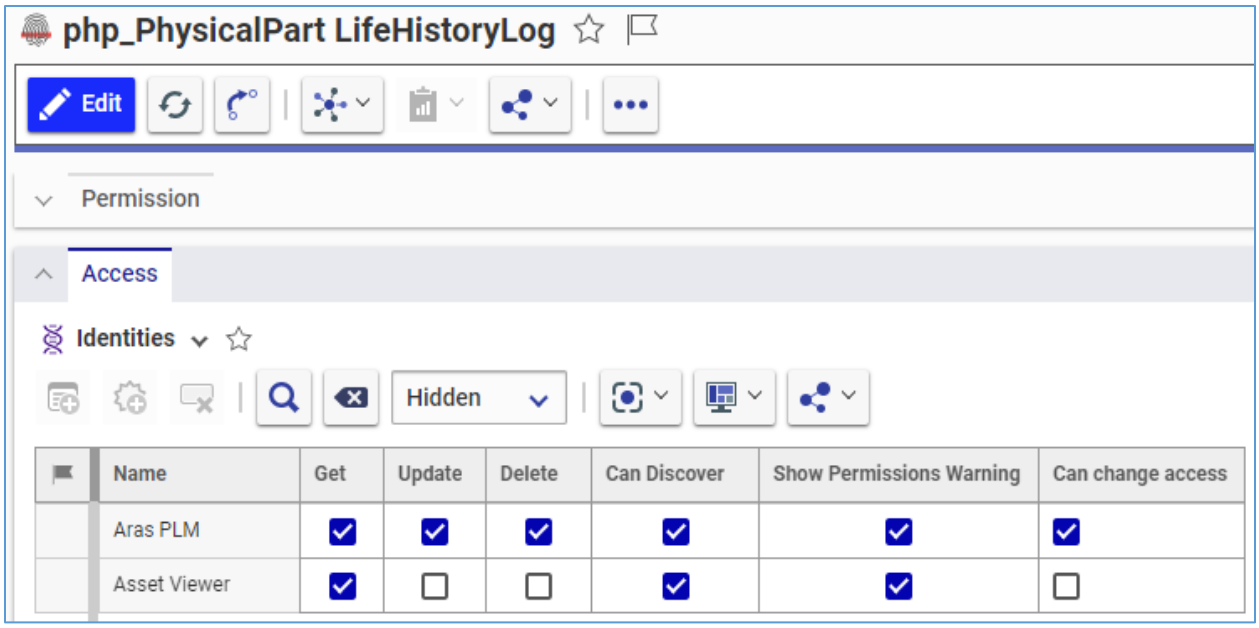



Figure 73.

The **PhysicalPart LifeHistoryLog** ItemType **Can Add** is set to the **Aras PLM** Identity. This setting should not be changed to allow users to create these Items.

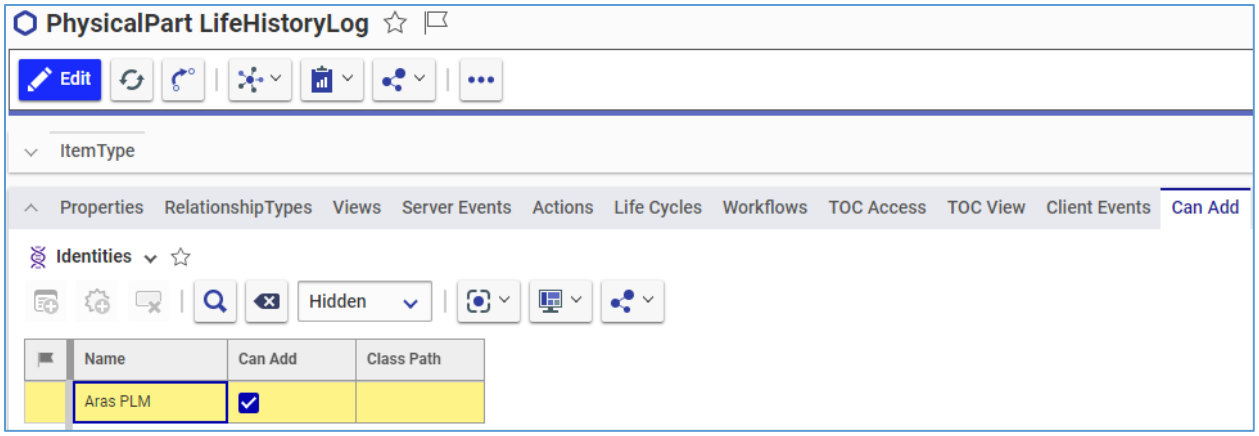



Figure 74.

# 16 OperationalEventType Items

The **OperationalEventType** Items are necessary only when tracking real-world operational activities of assets.

If your organization does not require tracking asset life variables by its operational events, you can delete or hide the **OperationalEventType** ItemType.

## 16.1 OperationalEventType Permissions

The default Permission for the **OperationalEventType** Items is **php_OperationalEventType**.

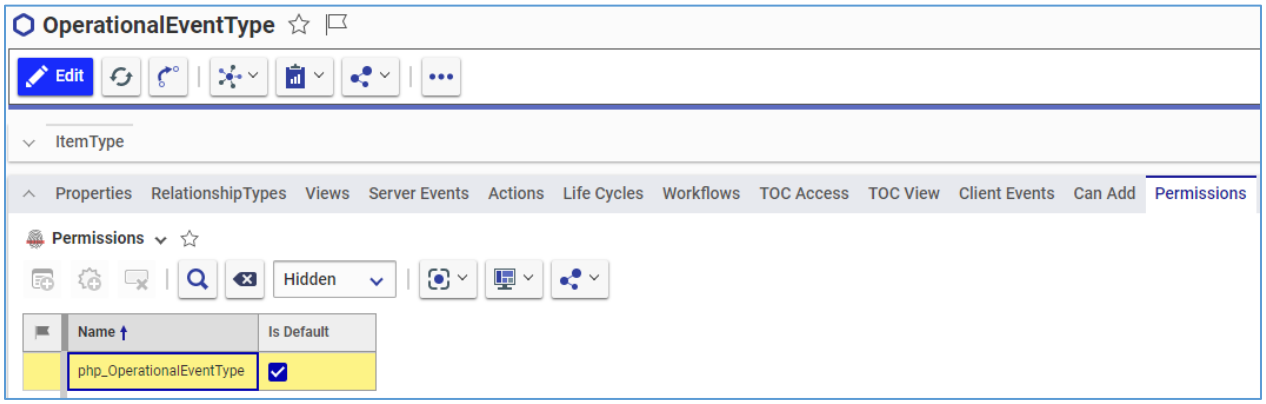



Figure 75.

It is the only Permission for the **OperationalEventType** Items.

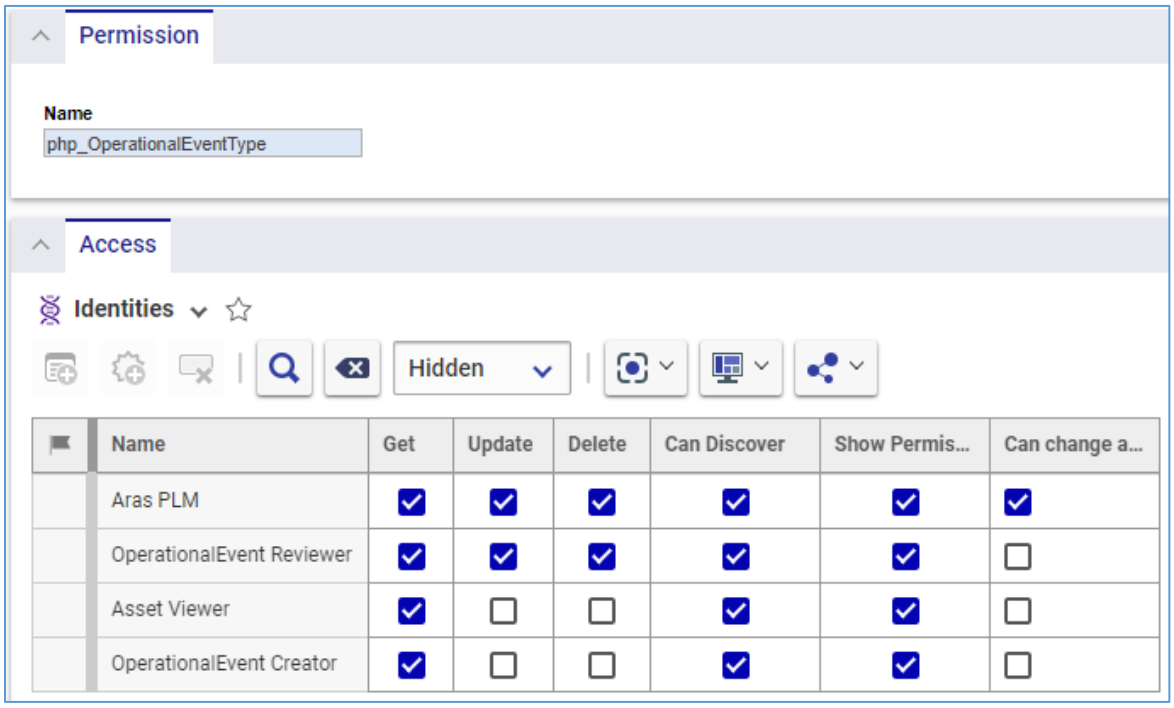



Figure 76.

The following table summarizes the Permissions for the **OperationalEventType** Items.

Table 5: The OperationalEventType Permissions matrix

| Can Add | Update | Delete | Can change access | Permission |
|---|---|---|---|---|
| OperationalEvent Reviewer, Aras PLM | OperationalEvent Reviewer, Aras PLM | OperationalEvent Reviewer, Aras PLM | Aras PLM | php_OperationalEventType |

**Note:** The **Aras PLM**, **OperationalEvent Reviewer, OperationalEvent Creator**, and **Asset Viewer** Identities have the **Get**, **Can Discover**, and **Show Permissions Warning** Access Rights for the **OperationalEventType** Items.

When updating the **Current Value** properties of the **PhysicalPart LifeValue** Relationship Items by the **Operational Event** Items, the **Asset Viewer** must have the **Get** and **Can Discover** Access Rights for the **OperationalEventType** Items. Having no such access, all Asset Identities cannot see foreign properties of the **PhysicalPart LifeValue** and **PhysicalPart LifeHistoryLog** Relationship Items sourced from **OperationalEventType** Items.

# 17 OperationalEvent Items

The **OperationalEvent** Items are necessary only when tracking real-world operational activities of assets.

If your organization does not require tracking asset life variables by its operational events, you can delete or hide the **OperationalEvent** ItemType.

## 17.1 OperationalEvent Permissions

The default Permission for the **OperationalEvent** Items is **php_PreliminaryOperationalEvent**.

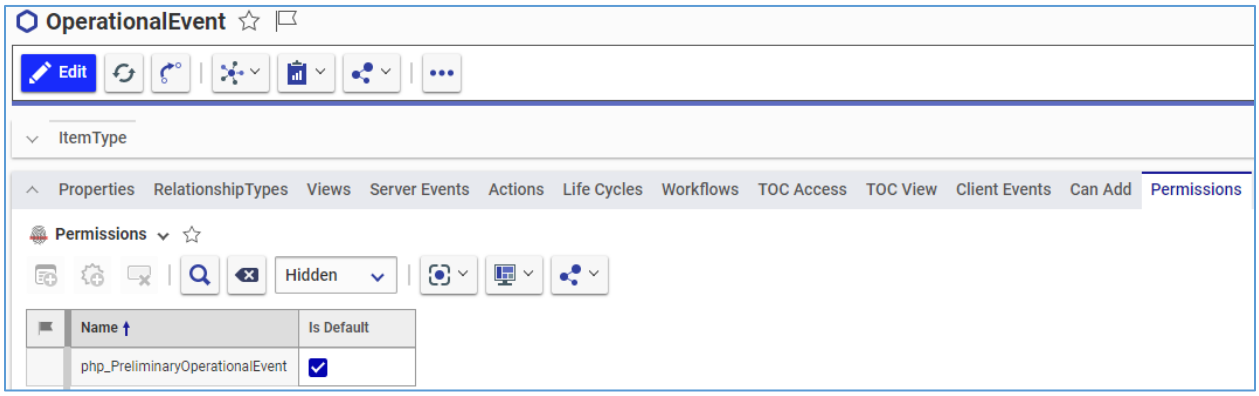



Figure 77.

The **OperationalEvent** Life Cycle Map overrides the default **php_PreliminaryOperationalEvent** Permission by setting another Permission in a particular Life Cycle State for an **OperationalEvent** Item. A newly created **OperationalEvent** Item is in the **Preliminary** State where it has the default Permission assigned.

The following table describes the **OperationalEvent** Life Cycle Permissions. The **Promoted by** column shows the Identities that can promote an Item to a given Life Cycle State. The **Permission** column refers to a Permission that manages Access Rights in a given Life Cycle State. The table uses the following abbreviations:

- OE C: the **OperationalEvent Creator** Identity
- APLM: the **Aras PLM** Identity
- AS V: the **Asset Viewer** Identity
- NA: not available
- OE R: the **OperationalEvent Reviewer** Identity
- OE A: the **OperationalEvent Admin** Identity

Table 6: The OperationalEvent Permissions matrix

| Life Cycle State | Can Add | Get | Update | Delete | Can Discover | Show Perm. Warning | Can change access | Promoted by | Permission |
|---|---|---|---|---|---|---|---|---|---|
| Preliminary | OE C, APLM | OE C, APLM, AS V | OE C, APLM | OE C, APLM | OE C, APLM, AS V | OE C, APLM, AS V | APLM | OE R | php_PreliminaryOperationalEvent |
| Review | NA | OE R, OE C, APLM, AS V | OE R, APLM | OE R, APLM | OE R, OE C, APLM, AS V | OE R, OE C, APLM, AS V | APLM | OE C | php_ReviewOperationalEvent |
| Complete | NA | OE A, OE R, OE C, APLM, AS V | OE A, APLM | APLM | OE A, OE R, OE C, APLM, AS V | OE A, OE R, OE C, APLM, AS V | APLM | OE R | php_CompleteOperationalEvent |

When updating the **Current Value** properties of the **PhysicalPart LifeValue** Relationship Items by the **Operational Event** Items, the **Asset Viewer** Identity must have the **Get** and **Can Discover** Access Rights for the **OperationalEvent** Items. Having no such access, all Asset Identities cannot see foreign properties of the **PhysicalPart LifeValue** and **PhysicalPart LifeHistoryLog** Relationship Items sourced from **OperationalEvent** Items.

# 18 OperationalEvent LifeUnit Items

The **OperationalEvent LifeUnit** Relationship Items are necessary only when tracking real-world operational activities of assets.

If your organization does not require tracking asset life variables by its operational events, you can delete or hide the **OperationalEvent LifeUnit** Relationship ItemType.

Because it is an automatically managed Relationship Item, an **OperationalEvent LifeUnit** Item is supposed to be created and deleted individually by the **Aras PLM** Identity that represents the system. A user should not be able to create or delete the **PhysicalPart LifeValue** Item because this may lead to data corruption when updating the **Current Value** properties of the **PhysicalPart LifeValue** Relationship Items by the **Operational Event** Items.

## 18.1 OperationalEvent LifeUnit Permissions

The default Permission for the **OperationalEvent LifeUnit** Items is **php_PreliminaryOperationalEventLifeUnit**.

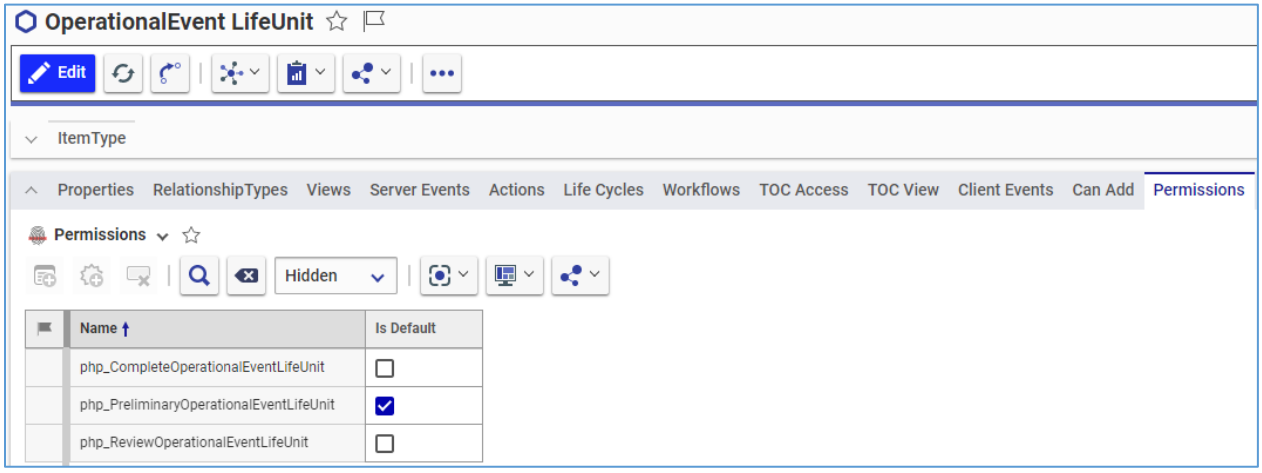



Figure 78.

The server side sets the current Permission for an **OperationalEvent LifeUnit** Item according to the current State of its source **OperationalEvent** Item. The **OperationalEvent** Life Cycle Map defines States for an **OperationalEvent** Item.

The following table describes the **OperationalEvent LifeUnit** Permissions. The **Permission** column refers to a Permission that manages Access Rights in a given Life Cycle State of a given source Item. The table uses the following abbreviations:

- OE C: the **OperationalEvent Creator** Identity
- APLM: the **Aras PLM** Identity
- AS V: the **Asset Viewer** Identity
- NA: not available
- OE R: the **OperationalEvent Reviewer** Identity

Table 7: The Physical Part Permissions matrix

| Source Item Life Cycle State | Can Add | Get | Update | Delete | Can Discover | Show Perm. Warning | Can change access | Permission |
|---|---|---|---|---|---|---|---|---|
| Preliminary | APLM | OE R, OE C, APLM, AS V | OE R, OE C, APLM | APLM | OE R, OE C, APLM, AS V | OE R, OE C, APLM, AS V | APLM | php_PreliminaryOperationalEventLifeUnit |
| Review | NA | OE R, OE C, APLM, AS V | OE R, APLM | APLM | OE R, OE C, APLM, AS V | OE R, OE C, APLM, AS V | APLM | php_ReviewOperationalEventLifeUnit |
| Complete | NA | OE R, OE C, APLM, AS V | APLM | APLM | OE R, OE C, APLM, AS V | OE R, OE C, APLM, AS V | APLM | php_CompleteOperationalEventLifeUnit |

When updating the **Current Value** properties of the **PhysicalPart LifeValue** Relationship Items by the **Operational Event** Items, the **Asset Viewer** Identity must have the **Get** and **Can Discover** Access Rights for the **OperationalEvent LifeUnit** Items. Having no such access, all Asset Identities cannot see foreign properties of the **PhysicalPart LifeValue** and **PhysicalPart LifeHistoryLog** Relationship Items sourced from **OperationalEvent LifeUnit** Items.

# 19 Appendix

## 19.1 Remove-and-Replace AML request examples

This section provides some examples of AML requests that perform various R&R operation cases. These examples should give you a general understanding of composing R&R AML requests and may not work if you just run them. For details about performing an R&R operation with AML, refer to section *7.2 Remove-and-Replace and AML*.

### 19.1.1 One-to-one replacement

```
<Item type="PhysicalPart" action="edit" id="2ABA7AB3CB474F5C9182861D24180C8D"
doGetItem="0">
    <is_remove_replace>1</ is_remove_replace>
    <Relationships>
        <Item type="PhysicalPart BOM" action="update"
id="A00162AF69DC4A488C466A545A035AA4" doGetItem="0">
            <end_on>2020-01-10T10:00:00</end_on>
            <replacing_id>55085AD1498A400090A649A2981D5109</replacing_id>
        </Item>
        <Item type="PhysicalPart BOM" action="add"
id="55085AD1498A400090A649A2981D5109" doGetItem="0">
            <source_id>2ABA7AB3CB474F5C9182861D24180C8D</source_id>
            <related_id>1868742AE8DC4283982A4F47FBF877F9</related_id>
            <is_inherit>1</is_inherit>
            <reference_designator>RD011</reference_designator>
            <start_on>2020-01-11T10:00:00</start_on>
            <replaced_id>A00162AF69DC4A488C466A545A035AA4</replaced_id>
            <quantity>1</quantity>
        </Item>
    </Relationships>
</Item>
```

### 19.1.2 Split

```
<Item type="PhysicalPart" action="edit" id="2ABA7AB3CB474F5C9182861D24180C8D"
doGetItem="0">
    <is_remove_replace>1</ is_remove_replace>
    <Relationships>
        <Item type="PhysicalPart BOM" action="update"
id="A00162AF69DC4A488C466A545A035AA4" doGetItem="0">
            <end_on>2020-01-10T10:00:00</end_on>
            <replacing_id>55085AD1498A400090A649A2981D5109,
A5085AD1498A400090A649A2981D5109</replacing_id>
```

```
        </Item>
        <Item type="PhysicalPart BOM" action="add"
id="55085AD1498A400090A649A2981D5109" doGetItem="0">
            <source_id>2ABA7AB3CB474F5C9182861D24180C8D</source_id>
            <related_id>1868742AE8DC4283982A4F47FBF877F9</related_id>
            <is_inherit>0</is_inherit>
            <reference_designator>RD011</reference_designator>
            <start_on>2020-01-11T10:00:00</start_on>
            <replaced_id>A00162AF69DC4A488C466A545A035AA4</replaced_id>
            <quantity>1</quantity>
        </Item>
        <Item type="PhysicalPart BOM" action="add"
id="A5085AD1498A400090A649A2981D5109" doGetItem="0">
            <source_id>2ABA7AB3CB474F5C9182861D24180C8D</source_id>
            <related_id>1868742AE8DC4283982A4F47FBF877F9</related_id>
            <is_inherit>0</is_inherit>
            <reference_designator>RD011</reference_designator>
            <start_on>2020-01-11T10:00:00</start_on>
            <replaced_id>B00162AF69DC4A488C466A545A035AA4</replaced_id>
            <quantity>1</quantity>
        </Item>
    </Relationships>
</Item>
```

### 19.1.3 Merge

```
<Item type="PhysicalPart" action="edit" id="2ABA7AB3CB474F5C9182861D24180C8D"
doGetItem="0">
    <is_remove_replace>1</ is_remove_replace>
    <Relationships>
        <Item type="PhysicalPart BOM" action="update"
id="A00162AF69DC4A488C466A545A035AA4" doGetItem="0">
            <end_on>2020-01-10T10:00:00</end_on>
            <replacing_id>55085AD1498A400090A649A2981D5109</replacing_id>
        </Item>
        <Item type="PhysicalPart BOM" action="update"
id="B00162AF69DC4A488C466A545A035AA4" doGetItem="0">
            <end_on>2020-01-10T10:00:00</end_on>
            <replacing_id>55085AD1498A400090A649A2981D5109</replacing_id>
        </Item>
        <Item type="PhysicalPart BOM" action="add"
id="55085AD1498A400090A649A2981D5109" doGetItem="0">
            <source_id>2ABA7AB3CB474F5C9182861D24180C8D</source_id>
```

```
            <related_id>1868742AE8DC4283982A4F47FBF877F9</related_id>
            <is_inherit>0</is_inherit>
            <reference_designator>RD011</reference_designator>
            <start_on>2020-01-11T10:00:00</start_on>

    <replaced_id>A00162AF69DC4A488C466A545A035AA4,B00162AF69DC4A488C466A545A035AA4</replaced_id>
            <quantity>1</quantity>
        </Item>
    </Relationships>
</Item>
```