

ACE
2024

STUDENT TRAINING GUIDE

Defining User Access with MAC Policies



Copyright © 2024 by Aras Corporation. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.

Distribution of the work or derivative of the work in any standard (paper) book form for a commercial purpose is prohibited unless prior permission is obtained from the copyright holder.

Aras Innovator, Aras, and the Aras Corp "A" logo are registered trademarks of Aras Corporation in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

Microsoft, Office, SQL Server, IIS, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Notice of Liability

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

Revision MARCH 2024



Overview

Aras Innovator gives administrators many options for access control. Mandatory Access Control (MAC) is designed to control access to Items by comparing characteristics of the Item being requested against those of the user requesting access.

For example, a MAC Policy can evaluate a User's properties (Clearance Level, Citizenship, Company, etc..) against a Part Item's properties (Classified/Non-classified, Confidentiality, Supplier, etc.).

In addition to properties directly on Itemtypes and users, attributes can be derived from related and/or referencing items and results used to determine access control policy.

For example, we may derive a list of security levels from all referencing Projects for a given Part. If that Part item is referenced by even one restricted project, access can be restricted to authorized users if needed.

Custom logic can also be used to determine environmental conditions, like time of day or other customer specific requirements. All these features empower administrators to implement a flexible and robust security scheme with MAC.

Course Goals

- Review Aras Access Control with emphasis on the way MAC fits into the overall scheme.
- Explore Mandatory Access Control as implemented using property expressions.
- Introduce the latest MAC feature "Multi-valued Derived Attributes".
- Finally, we will implement example MAC policies on the machines provided.

Contents

Part One:

- Brief Review - Innovator Access Control Concepts

Part Two:

- Mandatory Access Control (MAC)

Part Three:

- Latest MAC Functionality - Multivalued Derived Attributes

Also:

- Resources / More Information



Objectives

We will explore Mandatory Access Control in three parts:

- Review of Innovator Access Control concepts
 - Permissions
 - Role-based Teams
 - Domain Access Control (DAC)
- Introduction to Mandatory Access Control (MAC)
 - How MAC works, and how it fits into the Access Control stack
 - Attribute-driven Boolean expressions
- Enhanced Access Control capabilities with Multivalued Derived Attributes
 - Query-driven collection operations

Access Rights

- *Get, Update, Delete, Discover, Show Warnings, Change Access*
- **Access Rights are Integral to all Access Control Schemes**
 - **Permissions**
 - **Teams**
 - **Domain Access Control (DAC)**
 - **Mandatory Access Control (MAC)**

Name	Get	Update	Delete	Can Discover	Show Permissions Warning	Can change access
Aras PLM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Manager	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Owner	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Creator	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
All Employees	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

4 | © 2023 Aras

Review: Access Rights

A matrix of access rights is common to all forms of access control in Aras Innovator. Access rights include Get, Update, Delete, Discover, Show Permissions Warning and Can Change Access. These rights are enabled or disabled for Identities as defined in a matrix.

The most basic form is the Permission item shown above. All schemes use access rights as the mechanism for security policies – Permissions, Teams, Domain Access Control (DAC), and Mandatory Access Control (MAC).

Itemtype Permissions

- Assigned to an **Itemtype** Definition
- Managed by **Lifecycle**
- Enforces **Access Rights**

The screenshot shows the Aras Innovator interface. On the left, the 'ItemType' definition for 'Part' is displayed, showing properties like Name, History Template, Versioning, and Class Structure. On the right, a 'Permission - New Part (read only)' window is open, showing a matrix of permissions for roles like Aras PLM, Manager, Owner, and Creator across actions like Get, Upd., Del., Can Discover, Show Permis..., and Can change a... The matrix shows that 'Get' is checked for all roles, while other actions like 'Upd.' are only checked for Manager and Owner. Above the windows, a state transition diagram for 'Aras PLM' is shown, with states including Preliminary, In Review, Released, In Change, Superseded, and Obsolete, connected by arrows labeled with transitions like 'Manual Change' and 'Aras PLM'.

Permissions are initially set by the Itemtype definition, then commonly overridden by LifeCycle State changes. They can be defined with abstract Roles, which are resolved when the context item is associated with a Team item.

Try It ... View a Permission Item:

1. Open the Navigation Panel to view the TOC.
2. Navigate to Design->Parts and open any preliminary Part item for viewing
3. From the [...] More menu, select Permission->View
4. Note that access to the opened Part item is controlled by this matrix
5. Open any Part in State 'Released', and view the Permission
6. Note the differences in the Access matrix
7. Close all Parts

Role-Based Teams

- Teams enable Run-time Assignment of Identities to Roles
- A Team Item on the Document maps specific Identities into Access Rights by Role

Standard Permissions with Role-based Teams are Collectively Referred to as “Role-Based Access Control” (RBAC)

Name	Get	Update	Delete	Can Discover	Show Permissions Warning	Can change access
Admin Role	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Editor Role	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Owner	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Creator	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Observer Role	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Aras | © 2023 Aras

Role-based Teams enable many assignment patterns on a single permission matrix

Teams contain mappings from Roles to Identities. They work in tandem with Permissions defined with Roles instead of discreet Identities in the vertical axis.

All Itemtypes include an Item Property ‘Team’. If a Team Item is assigned to this Property, corresponding Roles are replaced with the specific Identity defined in the Team item.

Workflow Assignments are also resolved when a Team is assigned to the Controlled Item.

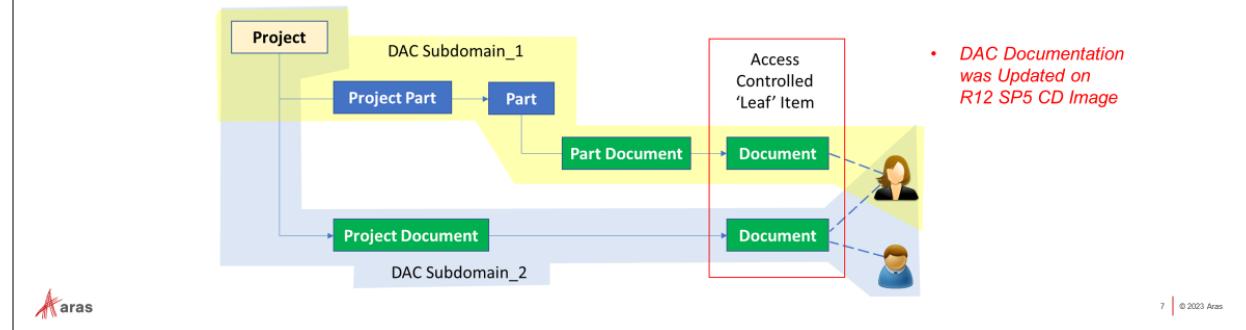
Try It ... View a Team Item:

1. Navigate to Administration->Teams in the TOC
2. Search/Open ‘Product Team’ and observe the mappings
3. Any Permissions with these Roles would replace them with the specified Identities:

Name	Role [...]
Terry Adams	Team Manager
Mike Miller	Team Member
Diane Prescott	Team Member

Domain Access Control (DAC)

- DAC determines which permission to grant based on two things:
 - 1) Item's **Relationship** to a root item – for instance a Project, Territory, Department or other "Domain"
 - 2) Matching criteria between Item and Root Item including LifeCycle State and Parent Permission
- Access Rights can be granted by DAC or RBAC



Access Control by Occurrence in a Relationship Structure

Domain Access Control (DAC) uses a standard Aras Query Definition item to determine the level of access to grant to an item. In the example above, Document Items can be (a) directly related to a Project or (b) related to a Part that is related to the Project. Because the structures are different, DAC can grant different permissions to Documents in one structure as opposed to the other.

Additional criteria like Lifecycle States and existing Permissions enable further refinement of Access granted to the user.

For more information:

Refer to the Domain Access Control Guide found in the Documentation Folder of the CD Image.

Mandatory Access Control (MAC)

- MAC Evaluates attributes of the Current User against attributes of the Current Item
- Boolean Expressions define the Conditions of Access
- If expressions solve TRUE, an Access Right is retained
- If FALSE, the Access Right is revoked



Aras

8 | © 2023 Aras

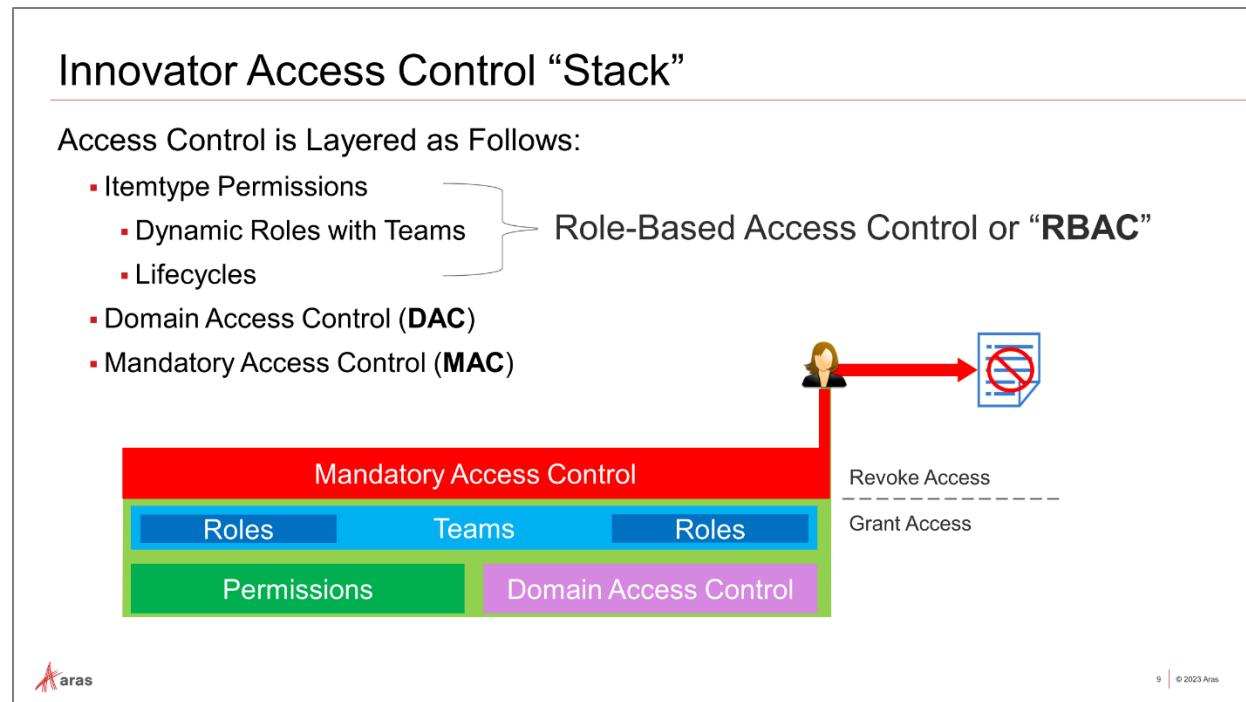
Mandatory Access Control

Unlike other Innovator Access Control models, MAC can revoke existing Access Rights instead of granting new Permissions. Therefore, it acts as a conditional override of any existing Access Rights for a given Item.

The decision on whether to revoke access or allow it is made by evaluating characteristics of the current User against the characteristics of the Item being accessed. MAC uses Boolean expressions to perform this test. If the Boolean expression solves to TRUE, then access is left as is. Otherwise, the Access Rights (Get, Update, Delete, etc.) are disallowed for the User requesting it.

Advanced Mandatory Access Control with Multi-valued Derived Attributes

To support advanced use cases MAC supports *Derived Attributes*, which are query-driven collections that can be used as operands in MAC Boolean expressions. This allows Access Control to be defined using set operations like intersection, containment, exclusion, etc. against Query results.



How Access Control is Layered

As previously mentioned, Permissions, Teams, and DAC grant Permissions to Items. Permissions are set by default on new Items using the default Permission template configured on the Itemtype. Permissions are often changed by Lifecycles. Role Identities can be overloaded dynamically by Team assignment. Domain Access can change Permissions on an item based on the structure where it occurs (and other criteria). All of these redefine or reassign Permissions.

MAC does not grant any Permissions – instead it enforces Boolean expressions against specific Access Rights (Get, Update, Delete, etc.). In this manner, MAC conditionally revokes existing rights.

All four of these Access Control mechanisms may be implemented for any given Itemtype. MAC is enforced last and has the final say on Access Rights.

Summary

- All Forms of Access Control enforce Access Rights
 - Get, Update, Delete, etc.
- Role-based Access Control (RBAC)
 - Itemtype Permissions, Teams Role resolution, LifeCycle State Permissions, etc.
- Domain Access Control (DAC)
 - Assigns Permissions based on Relationship structure of Item
- RBAC and DAC can **grant** Permissions to Items
- MAC can **deny** Access Rights granted by RBAC/DAC

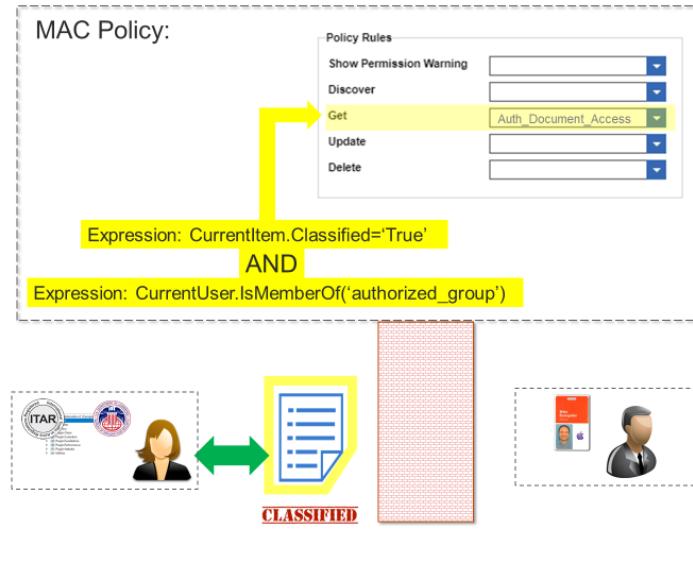


10 | © 2023 Aras



We will focus solely on MAC and Multi-valued Derived Attributes from this point forward.

How MAC Policies Work



Applying Conditions to Access Rights

Let's dig a little deeper into the simple example use case where only authorized users should be allowed to view (Get) classified Document items. MAC uses Boolean **expressions** in a structure that collectively solves to a single TRUE or FALSE condition – often made up of several nested expressions with logical AND/OR/NOT operators. The overall **Condition** is then applied to one or more Access Rights to define a **MAC Rule**.

- Any Property of CurrentItem or CurrentUser can be made available for use in expressions.
- Useful *Helper Functions* simplify expressions for example IsMemberOf() allows User qualification by adding or removing users from Identity groups.

Rule			
Expressions		Datatype	Logical
1	CurrentItem.Classified = 'True'	List Property [True, False]	AND
2	CurrentUser.IsMemberOf('authorized_group')	Group Identity	-

We could alternatively add properties to the User itemtype directly, but group membership avoids editing of the User Itemtype and is more maintainable. Below is an expression using a Property maintained directly on the User Itemtype, which would require editing and updating the item:

-	CurrentUser.[ITAR Level] = "3"	List Property on User Item	-
---	--------------------------------	----------------------------	---

MAC Policy Rules

- Conditions Applied to Innovator Access Rights define MAC Rules
 - Conditions Consist of Boolean Expressions
 - Boolean Expressions evaluate Properties on Current User and Current Items
 - If MAC Conditions are not met access right is denied regardless of RBAC, DAC rights

Access Rights

Aras

Examining a MAC Policy Rule

In the MAC data model, a **Rule** associates Boolean Conditions with one or more Access Rights. The Condition must solve TRUE or the Access Right is revoked for the controlled Item. MAC Rules may control any Access Right except for 'Can Change Access' (changing the underlying Permission is not applicable).

Try It ... View an existing MAC Rule:

1. In the TOC, navigate to Administration->Access Control->MAC Policies
2. Search for and view the 'Hide Templates' MAC policy
 - Note that this policy applies to the Document Itemtype (per the 'Applied To' tab)
 - Note that *Update* and *Delete* Access Rights have a Policy Rule assigned to them
3. From the sidebar menu, open the Rule Editor
4. Double-click on the Rule 'Diane Prescott and Terry Adams' to expand the Rule syntax:

The current Users' login names must equal 'dprescott' OR 'tadams'

Whenever the current Item has the 'Is Template' flag set to true (1)

OR true if not a template (retains existing right)

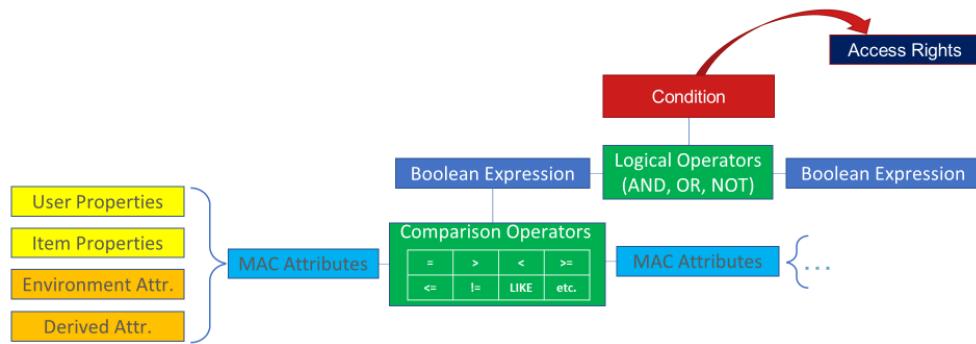
```

&  ||  ! | ✓  ✘
1  (
2   (
3     (CurrentUser.[Login Name] = 'dprescott')
4     OR (CurrentUser.[Login Name] = 'tadams')
5   )
6   AND (CurrentItem.[Is Template] = 1)
7 )
8 OR (CurrentItem.[Is Template] = 0)
9

```

Anatomy of a MAC Policy

- A Condition Applied to an Access Right forms a MAC RULE
- Conditions consist of Boolean Expressions
- Boolean Expressions consist of Attributes and Operators
- Attributes evaluate Properties of Items and Users



Aras

15 | © 2023 Aras

MAC Policy Structure

The Boolean logic defining a Condition can be a simple expression or a complex structure of expressions with User Properties, Item Properties, user-defined Environment Attributes, and data sets called Derived Attributes which ultimately solve to a single true/false value as a Rule Condition.

Resolve Boolean Expressions to a Single True/False Result

Each unit expression can be combined with Logical Operators (AND, OR NOT) to define Conditions. MAC Policies often evaluate attributes of the User requesting access in expressions with attributes of the (requested) Item to decide whether (or not) to revoke that User's access to the Item. Therefore, MAC Conditions often have the structure:

<User attribute expression> AND <Item attribute expression>

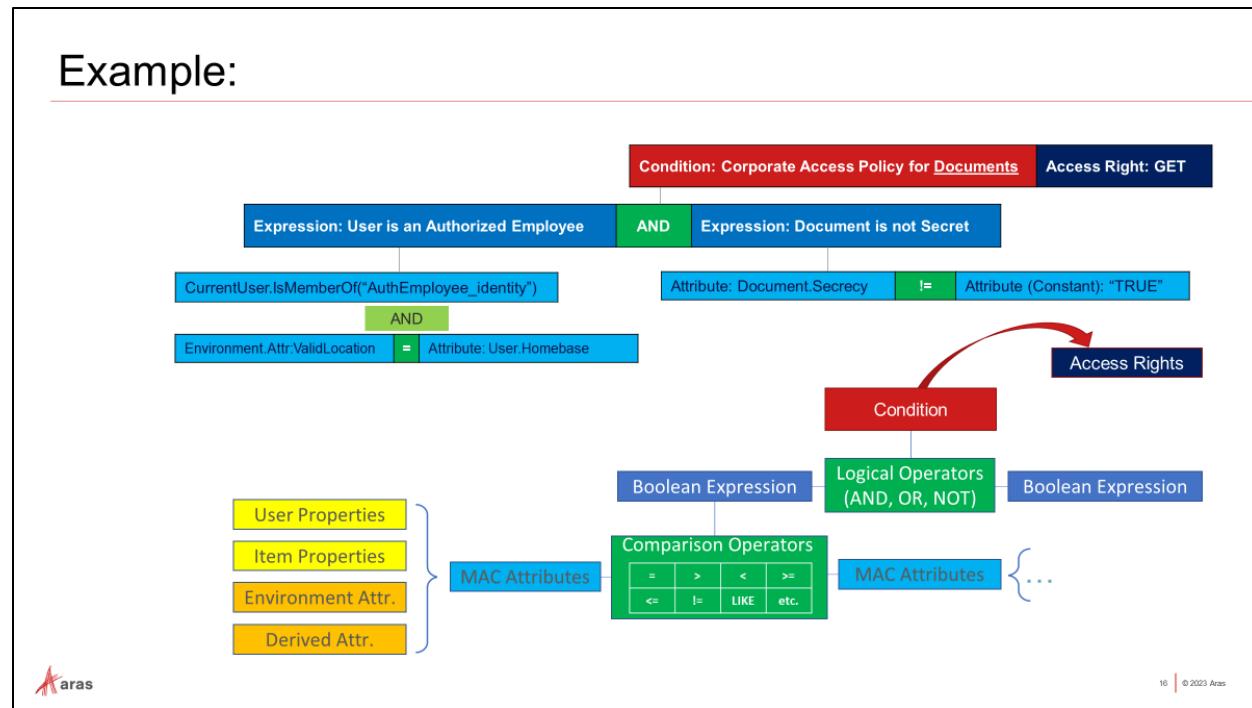
User attributes are referenced as **CurrentUser.<attribute>**, and Item attributes are accessed as **CurrentItem.<attribute>**:

- CurrentUser is always the current logged in user in the expression being evaluated in the Rule.
- CurrentItem is always the item being accessed, as configured in the MAC Policy under the "Applied To" tab.

For instance:

```

&  ||  !  ✓  ✗
1  (CurrentUser.[Company Name]='Aras' ) AND ((CurrentItem.state='Released'))
  
```



MAC Condition Hierarchy Example

The Attribute is the value(s) used in Boolean Expressions, whether a standalone Boolean Attribute or juxtaposed with others using Comparison Operators to determine a true/false result.

This unit expression can then be used as is or used in a nested expression structure to ultimately produce one single Boolean result. When fully defined, the Condition is applied to one or more Access Rights to determine whether or not to unilaterally revoke that right if the Condition is false.

(The slide automation demonstrates this progression)

MAC Attributes

- Attributes are the operands in Boolean expressions
- Attributes can be:
 - User Properties
 - Item Properties
 - Environment Attributes
 - Derived Attributes (covered later)

Access Rights → Condition → Boolean Expression → Logical Operators (AND, OR, NOT) → Boolean Expression → Comparison Operators → MAC Attributes
 MAC Attributes ← {User Properties, Item Properties, Environment Attr., Derived Attr.} → ...

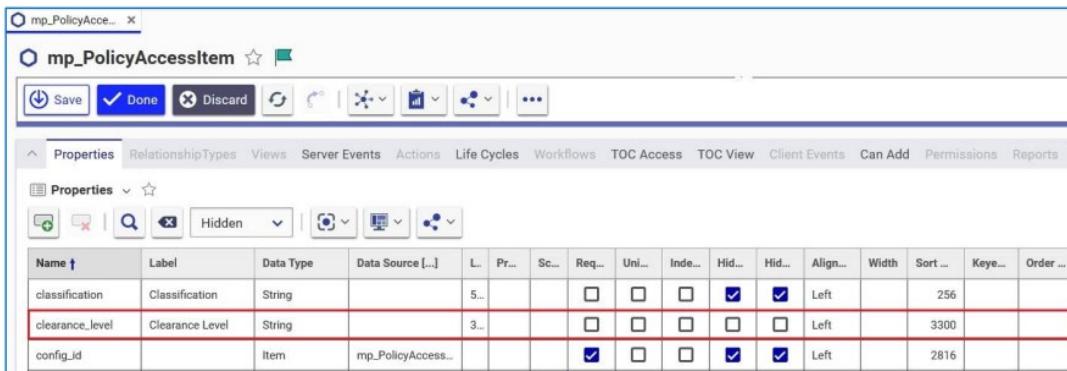
MAC Attributes

In MAC the term attribute is a general abstraction for any value or set of values that can be used within a boolean expression. They may be single-valued, or multi-valued, and used interchangeably where type and cardinality allow. MAC Attributes may be any one of the following:

- A Constant of the correct datatype, cardinality, and value.
- A Property defined on the current Itemtype being accessed.
- A Property on the User Itemtype for the current user making the access request.
- Environment Attribute dynamically set by execution of a (custom) Method.
- Derived Multi-valued Attribute produced from the results of a Query Definition.

Adding Custom Itemtype Properties as Supported MAC Attributes

Only custom properties added to the **mp_PolicyAccessItem** ItemType can be referenced in a MAC expression. To add custom properties edit the **mp_PolicyAccessItem** ItemType and add any properties that you wish to reference as Attributes - e.g., *clearance_level* below:



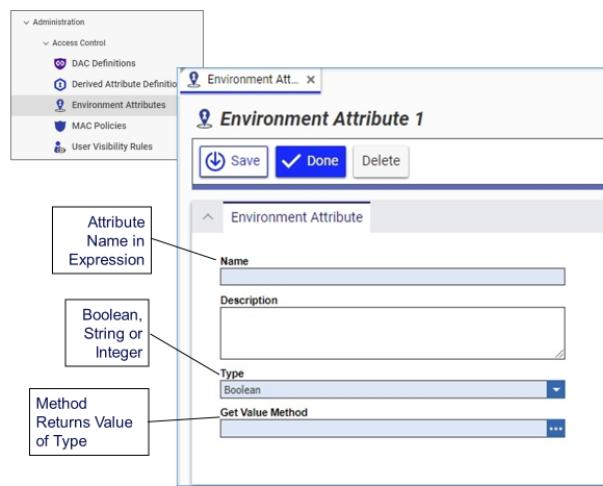
MAC Environment Attributes

Define your own Custom Attributes

- Can return Boolean, string or integer value to expression
- Name becomes symbol
- Method logic generates type value when referenced in MAC Expression



Refer to Section 2.2 in the MAC Policies Documentation



Aras

22 | © 2023 Aras

Method Based Environment Attributes

Attributes for MAC Expressions can be defined as return values of methods using the MAC Environment Attribute. The item is given a Name, which can be referenced in MAC expressions. A Method is provided, to compute a value which may be:

- Boolean
- String or
- Integer

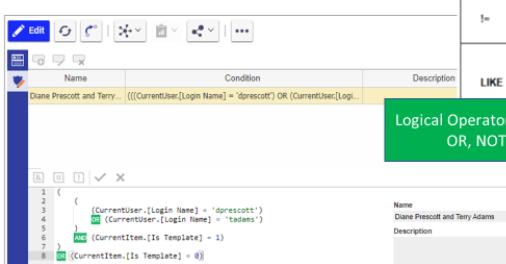
The value is then resolved by Method logic to serve as an Attribute in MAC expressions. The example in the MAC documentation provides an example where a Boolean attribute determined true if the request is being made during “work hours” as defined the Method code provided in the appendix.

This approach allows for high levels of flexibility in definition of Attributes that are not necessarily based on User or Item properties.

Expression Editor

Rules Item includes an Expression Editor

- CurrentItem; CurrentUser Variables
- Comparison Operators 
- Logical AND, OR, NOT Operators
- Built-in Syntax Checking and Code Completion



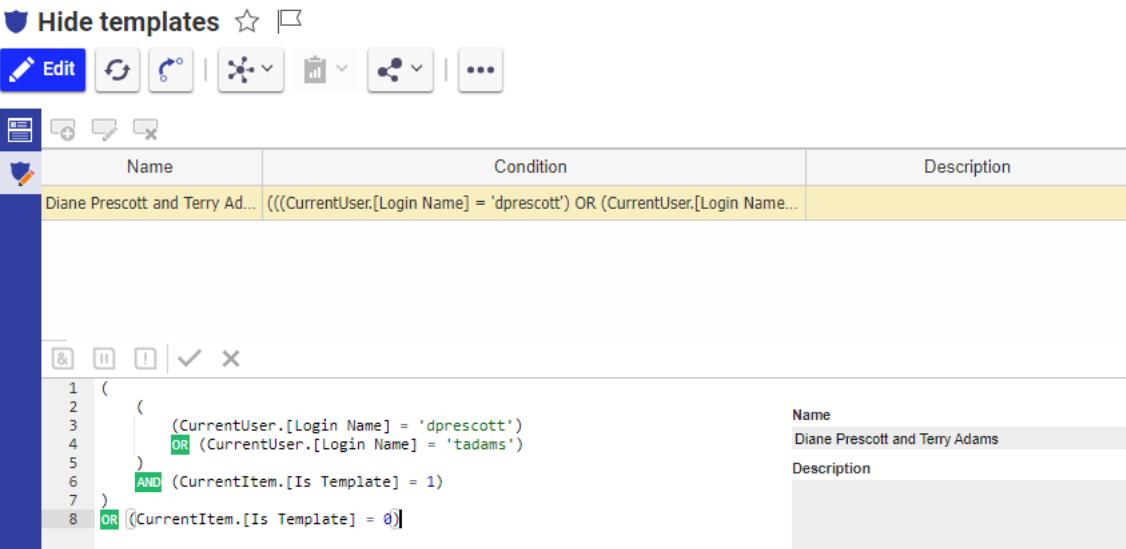
Operation	Name	Usage	Meaning
=	Equals	valueRef1 = valueRef2	TRUE if left value is equal to right value.
>	Greater Than	valueRef1 > valueRef2	TRUE if left value is greater than right value.
<	Less Than	valueRef1 < valueRef2	TRUE if left value is less than right value.
>=	Greater Than or Equal To	valueRef1 >= valueRef2	TRUE if left value is greater than or equal to right value.
<=	Less Than or Equal To	valueRef1 <= valueRef2	TRUE if left value is less than or equal to right value.
!=	Not Equal To	valueRef1 != valueRef2	TRUE if left value is not equal to right value.
LIKE	Like	valueRef1 LIKE valueRef2	True is left value matches the right value (pattern). The syntax for the "LIKE" operator is exactly the same as in Transact-SQL.

 17 | © 2023 Aras

Using the Rule/Expression Editor

The Expression Editor simplifies the definition of MAC Conditions. CurrentItem, CurrentUser Attributes and Environment Attributes are available for use in Expressions. These can be interactively combined with Logical Operators (AND, OR NOT) to collectively produce a single TRUE/FALSE result – i.e., Condition.

While examining an existing MAC Policy in an earlier exercise we used the Expression Editor from the sidebar menu of the **Hide Templates** MAC Policy. We will review the Editor in detail and use it to define a new Policy in upcoming exercises.



The screenshot shows the Expression Editor interface for a MAC Policy named "Hide templates". The editor displays the following code:

```

1  (
2    (
3      (CurrentUser.[Login Name] = 'dprescott')
4      OR (CurrentUser.[Login Name] = 'tadams')
5    )
6    AND (CurrentItem.[Is Template] = 1)
7  )
8 OR (CurrentItem.[Is Template] = 0)

```

The sidebar on the left shows the policy details:

- Name:** Diane Prescott and Terry Adams
- Description:**

Helper Methods

General

Method	Response
<code>CurrentUser.IsMemberOf(<Identity Name>)</code>	Returns true if the current user is a member of a (non-system) Identity <code><Identity Name></code> , otherwise it returns false.
<code>CurrentUser.IsMemberOf(Property<Item>)</code>	Returns true if the current user is a member of the Item Property of type Identity. For example: <code>CurrentUser.IsMemberOf(CurrentItem.identity_id)</code>
<code>String.Contains(<StringToSearch>, <SearchForString>)</code>	Returns true if <code><SearchForString></code> is a substring of <code><StringToSearch></code> , otherwise it returns false.
<code>CurrentItem.HasUserVisibilityPolicyAccess()</code>	Returns true if the current user has access to the current item based on the active User Visibility Rules. This function can only be applied to User, Alias, Identity Item Types.

Extended Classification

Method	Response
<code>CurrentItem.IsXPropertyDefined((<xPropertyName>))</code>	Returns true only when the <code><xPropertyName></code> is defined on CurrentItem.
<code>CurrentItem.IsClassifiedByXClass(ss(<xClassName>))</code>	Returns true only when CurrentItem is classified by <code><xClassName></code> .
<code>CurrentUser.IsXPropertyDefined((<xPropertyName>))</code>	Returns true only when <code><xPropertyName></code> is defined on Current.User.
<code>CurrentUser.IsClassifiedByXClass(ss(<xClassName>))</code>	Returns true only when CurrentUser is classified by <code><xClassName></code> .



Built-in Helper Methods Simplify Expression Logic

The Expression Editor provides Helper Methods to simplify User, Item and String evaluation as detailed below:

<code>CurrentUser.IsMemberOf(<Identity Name>)</code>	Returns true if the current user is a member of a (non-system) Identity <code><Identity Name></code> , otherwise it returns false.
<code>CurrentUser.IsMemberOf(Property<Item>)</code>	Returns true if the current user is a member of the Item Property of type Identity. For example: <code>CurrentUser.IsMemberOf(CurrentItem.identity_id)</code>
<code>String.Contains (<StringToSearch>, <SearchForString>)</code>	Returns true if <code><SearchForString></code> is a substring of <code><StringToSearch></code> , otherwise it returns false.
<code>CurrentItem. HasUserVisibilityPolicyAccess()</code>	Returns true if the current user has access to the current item based on active User Visibility Rules. This function can only be applied to User, Alias, Identity Item Types.

Note: For Extended Classification XClass Methods `IsXPropertyDefined()`, `IsClassifiedByXClass()` are available for `CurrentItem` and `CurrentUser`. Refer to MAC Policies Documentation for more information.

Create a MAC Policy

- MAC Policy Enforces Conditions on Access Rights
- Conditions contain Expressions, with built-in abstractions for Current User, Current Item
- Syntax is Checked Interactively

The screenshot shows the Aras interface for creating a MAC Policy. The main window displays a policy named 'Example Mac Policy' with several rules listed. A red arrow points from the 'Conditions' section of the slide to the 'Condition01' row in the policy rules grid. A callout box labeled 'Boolean Expressions' points to the condition row. Another callout box labeled 'User Properties' and 'Item Properties' points to the properties listed below the condition row.

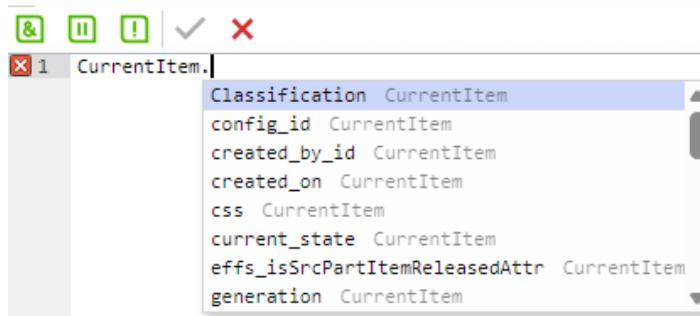
Creating a MAC Policy

In this exercise, we will work together to define a new MAC Policy, activate it, and observe its behavior. When activated, the policy will override the normally acceptable access to Documents even if permitted by the Permission in place. It configures this constraint to only certain group members as well.

The business use case is to prevent Asset Editors from being able to access Preliminary Documents.

Try It ... A Simple MAC Policy Based on Item LifeCycle

1. Navigate to Administration->Access Control->MAC Policies and create a new MAC Policy named 'Test Policy'
2. Under the 'Applied To' tab, add a reference to Document Itemtype, and Save the policy
3. This displays a new sidebar button for the Expression Editor Pane. Open the Editor.
4. Use the 'New Condition' toolbar button to add a new row to the Rules grid. You will now be able to type in the lower pane to define expressions.
5. Enter 'CurrentItem.' (note period) to see the available Properties for use in the Expression



6. Select 'CurrentItem.State' then type '='Preliminary'
7. Use spacebar to display Operators, choose AND
8. Add another space, then type NOT
9. Next select 'CurrentUser.IsMemberOf('Asset Editor')'. The expression should look like this:

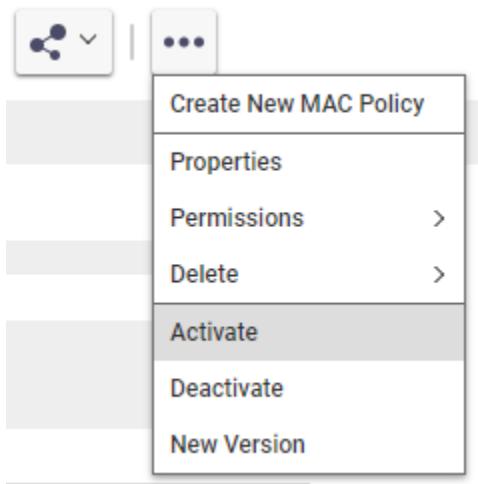
$$(\text{CurrentItem.state} = \text{'Preliminary'}) \text{ AND } (\text{NOT CurrentUser.IsMemberOf('Asset Editor')})$$
10. Next add an OR expression to allow access to Documents that are not in review:

```

1 (
2   (CurrentItem.state = 'Preliminary')
3     AND NOT (CurrentUser.IsMemberOf('Asset Editor'))
4 )
5 OR NOT ((CurrentItem.state = 'Preliminary'))

```

11. In the Name field enter 'Asset Editor Access'
12. The green checkmark indicates that there are no syntax errors.
13. Save the MAC Policy.
Note: the Rule syntax will be reformatted as shown above after saving.
14. Return to the main MAC Policy form, and add the new condition to **Get** and **Discover** Access Rights
15. Click Done, and use the More Toolbar command to Activate the Policy



16. You may get a pop-up indicating that more than one user is logged in, click OK
17. Log out and log in as mmiller (Mike Miller) who is an Asset Editor.
18. List Documents. Try to list Documents that are in state 'Preliminary' using the search bar.
19. Deactivate the Policy as Admin and login once again as Mike Miller. List Documents with state 'Preliminary' again.

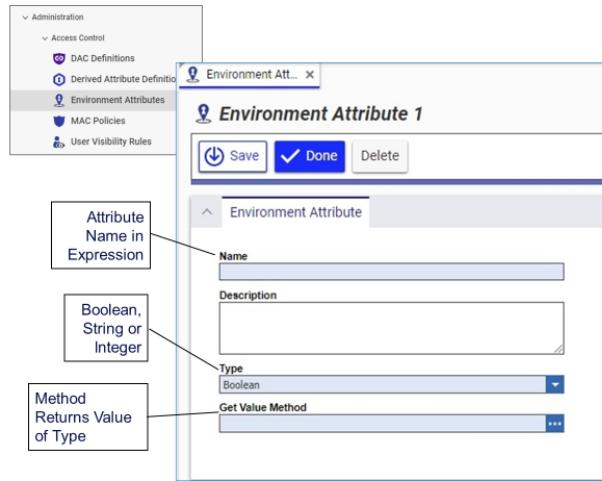
 Despite the Permissions assigned to Documents by the LifeCycle, MAC conditionally overrides access for members of the group when active.

MAC Environment Attributes

Define your own Custom Attributes

- Can return Boolean, string or integer value to expression
- Name becomes symbol
- Method logic generates type value when referenced in MAC Expression

- Refer to Section 2.2 in the MAC Policies Documentation



Sample Environment Attribute Method

Environment Attributes allow values used in expressions to be defined using custom Method logic rather than Property values. The following example shows how to write a method 'isWorkHours' that returns the value 'true' from 8AM to 8PM server time, Monday through Friday, otherwise 'false':

```
//MethodTemplateName=CSharp:Aras.Server.Core.AccessControl.EnvironmentAttributeMethod;
var startWorkTime = new TimeSpan(8, 0, 0); var endWorkTime = new TimeSpan(20, 0, 0);
var currentDate = DateTime.Now; var isWorkDay = DayOfWeek.Monday <=
currentDate.DayOfWeek && currentDate.DayOfWeek <= DayOfWeek.Friday;
var isWorkTime = startWorkTime <= currentDate.TimeOfDay && currentDate.TimeOfDay <=
endWorkTime; var isWorkHours = isWorkDay && isWorkTime;
attribute.SetValue(isWorkHours);
```

This Environment Attribute 'work_hours' could be referenced in a Rule Expression by Name:

Name	work_hours
Description	Returns True if the server time is between 8AM and 8PM, otherwise returns false.
Type	Boolean
Get Value Method	isWorkHours

Summary

- While RBAC and DAC Grant Access, MAC Conditionally Revokes it
- A MAC Policy consists of:
 - Policy **Rules** against Access Rights
 - ...consisting of **Conditions** with Logical Operators (AND, OR, NOT)
 - ...which consist of Boolean **Expressions** with Comparison Operators
 - ... Expressions use **Attributes**
 - Attributes for MAC are obtained from Properties found directly on requesting User and Item being requested, or by Method logic, or Derived by Query*
 - Custom Properties used as MAC Attributes must be ‘registered’ in mp_PolicyAccessItem



Summary Thus Far

We have learned that MAC is the final layer of the Innovator Access Control ‘Stack’, that is to say that regardless of any established access MAC can remove the Access Right that a MAC Rule is assigned for ultimate control. MAC revokes existing access, it does not re-assign permission (potentially granting access) like other policies may.

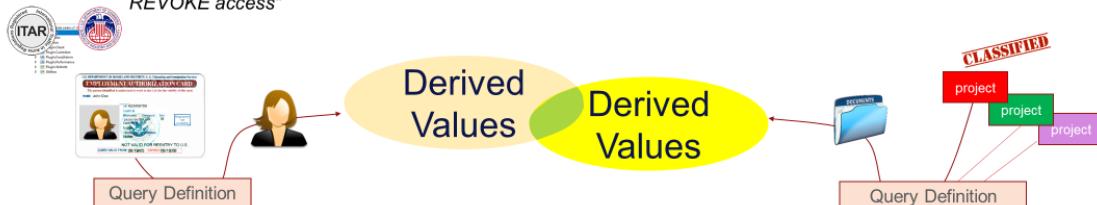
Rules are Boolean expressions evaluating Attributes. Attributes may be:

- Property values on Items,
- Property values on Users,
- Dynamically generated values via Method code (Environment Attributes),
- They may also be Collections derived from Query Definitions, called **Multivalued Derived Attributes**

Multivalued Derived Attributes will be the focus of the remainder of this course.

Introducing MAC Derived Attributes

- **MAC Derived Attributes** were released in R12 SP6+
- Like regular MAC Attributes, they are used to define policy Rules...
- Derived Attributes offer powerful new features:
 1. Property Values from **Related or Referencing Items** can also be used in expressions
 - E.g. "Check if the Projects Referencing this Item Require Extra Security"
 2. Rather than single Property values, multi-valued sets or **collections** are supported
 - E.g. "If [all User's qualifications] vs. [all qualifications required by Projects where used] has no matches then REVOKE access"



Aras

23 | © 2023 Aras

Attributes Can be Multi-valued Collections Derived by Query Definition

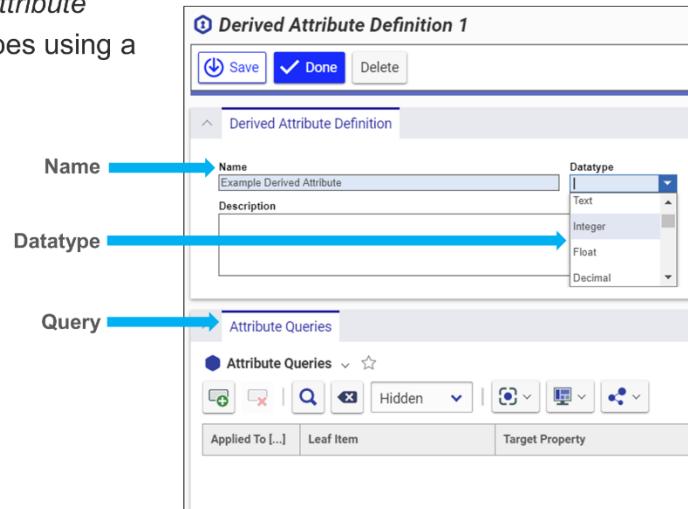
Multi-valued Derived Attributes have been added to MAC Policies allowing attributes to be collections (sets) of values derived from Query Definitions. This provides an extremely flexible way to form Boolean expressions to implement logic to control access to data.

For example, a query could derive all ITAR levels that a User has been added to, provided the ITAR level was a container Item with a relationship to User. Another could derive all Programs where a specific Customer was involved, provided the Customer item was assigned to the item property on the Program item. These Collections can be evaluated in Expressions using set operations like 'Overlaps', 'Contains', 'IsEmpty' as shown below:

Operator	Description
<code>Collection.IsEmpty(<multival attribute>)</code>	TRUE if Collection is empty
<code>Collection.Contains(<multival attribute>, value)</code>	TRUE if Collection <multival attribute> contains the value <value> (type
<code>Collection.Overlaps(<multival attribute1>, <multival attribute2>)</code>	TRUE if Collection <multival attribute1> has values in common with <multival attribute2>

A new Itemtype was added for *Derived Attributes*

- Defines *Multivalued Attribute Collections* for Itemtypes using a Query Definition



27 | © 2023 Aras

Derived Attribute Definition Itemtype

Derived Attributes are created and managed using Derived Attribute Definitions Items. The fields are completed as follows:

- Name becomes the symbol used to reference the Derived Attribute in Expressions
- The Collection has a specific Datatype, i.e., the type of query results

Queries are defined under the Attribute Queries tab.

- “Applied To” Itemtype (root & scope)
- The second column is double-clicked to open the Query Definition form
 - o The Query must result in a Target Property of the Leaf Item in the query structure
- Target Property will appear in the (read only) third column when the Query is defined. Type must match ‘Datatype’

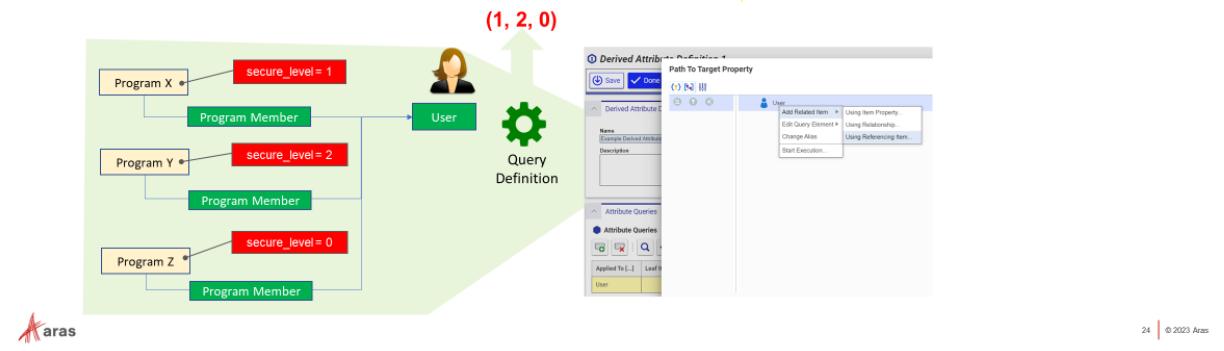
Derived Attributes Use Query Definitions

- An Integrated Query Specifies a “Path” to a Related/Referencing Item, and the Property to use as MAC Attribute

For example:

- “Query for the Set of all Security Level Properties from all **Programs** that the **Current User** is Assigned to”

CurrentUser.ProgramSecure_levels ← Used in MAC Boolean Expressions



Generating an Attribute as a Collection

In the example shown by the diagram above, the user is a member of three Programs. That is to say that Program Itemtype has a Relationship to the User Itemtype, and this User has a Relationship to three Program instances.

The query language used to show data in a Tree Grid View would use the same construct to derive target property values from the parent Program. In fact, one could test the logic by executing a Query Definition with the same query.

When saved, the name of the Derived Attribute can be referenced in a MAC Rule Expression as a Collection Attribute.

For example:

Collection.Overlaps (

CurrentUser.<derived_attr name1>,

CurrentItem<derived attr name2>

)

Derived Attributes can generate Collections in a manner similar to what we do for TGV content, with some restrictions. The query must generate values consistent with the ‘Datatype’ setting and compatible with other attributes and operators.

Performance considerations

Long running queries will impact performance, so queries should be kept concise and the data model against which they run should be evaluated for impact.

Attribute Queries

- Applied To (itemtype)** – the itemtype scope for the Derived Attribute. Also root of path.
- Leaf Item** – Related/Referencing Item containing Target Property. Double-click cell to access **Query Definition**.
- Target Property** – Property on the Leaf Item at the end of the Query Path whose value is used in the Collection
 - May be multi-valued if many related/references, or single-valued

The screenshot shows the Aras interface. At the top is the 'Derived Attribute Definition' dialog with fields for Name (Example Derived Attribute), Datatype (Integer), and Description. Below it is the 'Path To Target Property' dialog, which displays a hierarchical path from a 'User' item down through 'Program Member' relations to 'Program' objects (Program X, Program Y, Program Z). Specific properties are highlighted: 'secure_level = 1' for Program X, 'secure_level = 2' for Program Y, and 'secure_level = 0' for Program Z. A 'Query Definition' icon is shown on the right. The Aras logo is in the bottom left corner, and the copyright notice '27 | © 2023 Aras' is in the bottom right.

Query Definition for Derived Attributes

The Attribute Queries tab provided with the Derived Attribute form provides the interface for defining multivalued collections for MAC rules using Query Definition. The three columns are described above, and columns 2-3 are populated when the Query Definition is completed.

Applied To [...]	Leaf Item	Target Property
User	Program User	source_id

The Query Definition window is accessed by double-clicking on the 'Leaf Item' column.

Try It ... Derived Attribute to get Programs the Current User is a Member of:

1. From Administration->Itemtypes open **Program** for editing
2. Add a new relationship to **User** and click Done

Tab O...	Relationship ... ↑1	Tab Label	Name ↑2	Description [...]
1	Program Project	Projects	Project	
2	Program User	Members	User	

3. Navigate to Administration->Access Control->Derived Attribute Definitions and create a new Derived Attribute
 - Assign Name as '**User Programs**'
 - Set Datatype to '**Item**'
 - Add a new Attribute Query and assign the Applied To Itemtype (column 1) to '**User**'
 - *Double click* on column #2 Leaf Item to open the Query Editor
 - Define a query as follows:
 - Add Related Item > Using Referencing Item... **Program User**

4. From the Program User row, select (target) Property **source_id**

source_id
Add Property
config_id
created_by_id
current_state
id
locked_by_id
managed_by_id
modified_by_id
owned_by_id
permission_id
related_id
source_id
Team

5. Click OK to close the query window
6. Save the Derived Attribute

The Derived Attribute 'User Programs' can now be used in Rule Expressions. We will use this Derived Attribute in MAC rules shortly.

The screenshot shows the 'User Programs' configuration interface. At the top, there is a toolbar with various icons for editing, saving, and sharing. Below the toolbar, the 'Derived Attribute Definition' section is open, showing a table with one row:

Name	Datatype
User Programs	Item

Below this, the 'Attribute Queries' section is open, showing a table with three columns:

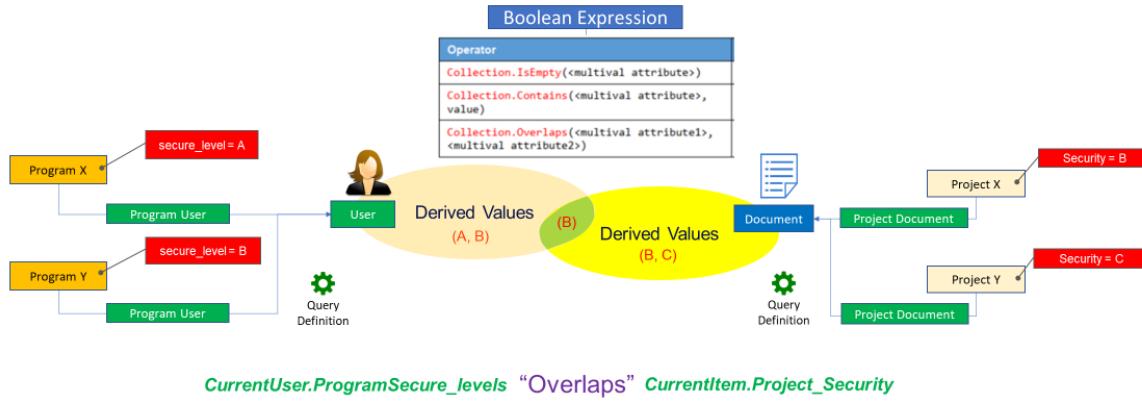
Applied To [...]	Leaf Item	Target Property
User	Program User	source_id

Note:

If desired, you can use a standard query via Administration->Configuration->Query Definition to duplicate this query and test results against the database. Keep in mind that Derived Attributes only use the Target property value in resulting Attribute Collections.

Derived Attributes in Boolean Expressions

- Derived Attributes are used in expressions similar to regular MAC Attributes
- New Operators are available for Collections



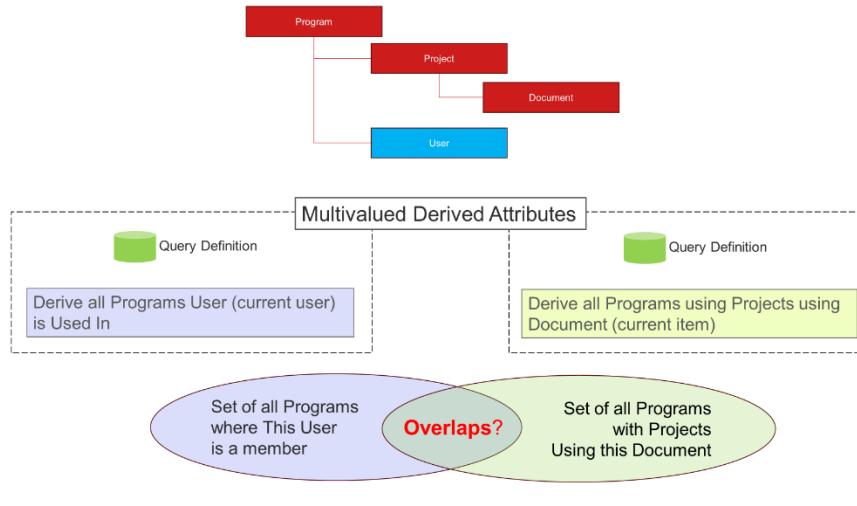
Using Derived Attributes in MAC Policy Rule Expressions

At this point we have created a multivalued attribute that dynamically returns all Programs that a User is a member of. We now need to define another Derived Attribute to obtain the list of Projects that the Document is used in.

The query should return all Program parents of Projects where the current Document is used – that is to say, where the Document being accessed is used as a Deliverable on a Project under the Program. This attribute will then be used in a Boolean Rule expression to determine whether a User is assigned to the same Program where the Document being accessed is used – or not. If not, then any existing access is revoked.

Use Case

Only Users related to a Program can access Documents on Projects related to that Program



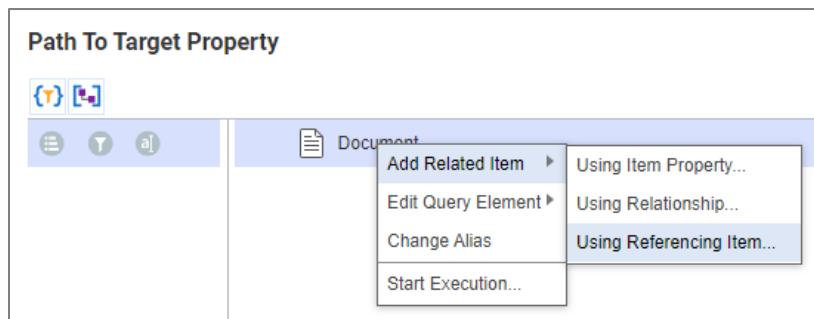
31 | ©2023 Aras

Try It ... New Derived Attribute to get Programs containing Project Documents:

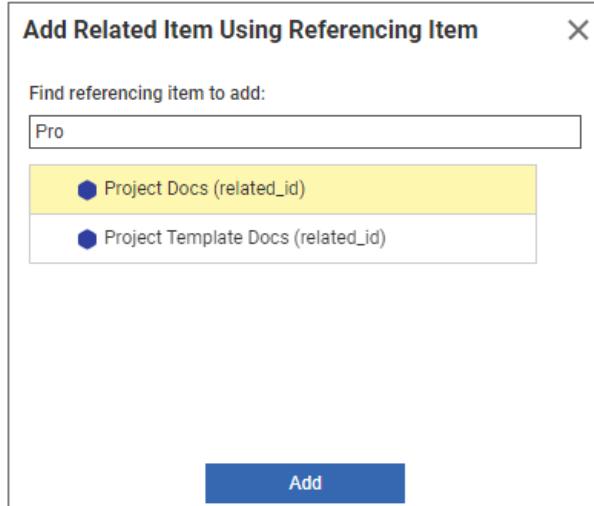
Navigate to Administration->Access Control->Derived Attribute Definitions

Create a second Derived Attribute:

1. Name: '**Document Project Programs**'
2. Datatype: **Item**
3. Add a new row in the 'Attribute Queries' tab
4. Enter '**Document**' in the Applied To [...] field
5. Double click on column #2 to open the Query Definition editor.
6. Define the query as follows:
 - o (right mouse menu) Add Related Item > Using Referencing Item...



- Document <-Referenced by **Project Docs**

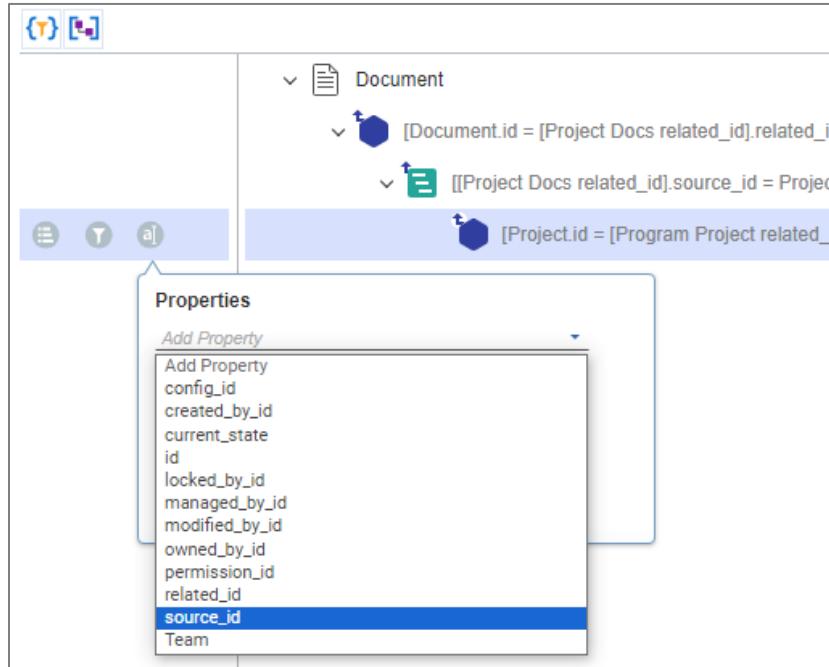


- With the new row selected:
Add Related Item > Using Item Property (**source_id**)

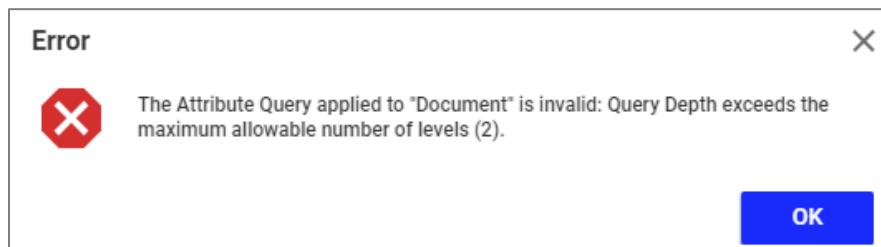
- To the third row: Add Related Item Using: Referencing Item “**Program Project**”

Defining User Access With MAC Policies

- With the fourth row selected, Select Property '**source_id**'
- Save and hit OK



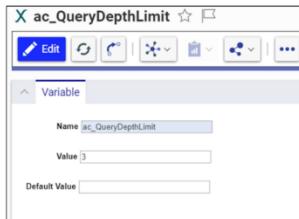
- Click OK to save the Query Definition
- Attempt to save the Derived Attribute 'Document Project Programs'
- You will get the following error:



- Keep the Derived Attribute window open, and proceed to the next page...

Configuring Depth Limit of Path Query

- By Default, the Depth of the Query Definition is restricted to 2 levels
 - This is to mitigate potential performance issues for extremely large structures with depth and breadth
 - Support can increase this level using an environment variable
- Variable: “**ac_QueryDepthLimit**”
 - If you need to increase the depth for your customer use case, you may do so by adding a Variable (Administration/Variables) named ‘ac_QueryDepthLimit’ as shown below.



WARNING: May cause performance issues.
Changes to this variable should be only
made with full consideration and testing of
results. Contact Aras Support for more
information.



29 | © 2023 Aras

Increasing the Depth Limit After Careful Examination

The system limits the depth of query levels by default to 2, which disallows attribute queries of more than 2 levels. Because Documents are Deliverables on Projects (under the Program) one of the queries must traverse the additional level to reach the parent Program.

In this case, we know that the number of Programs is low and will remain so, and the Project within each Program are few. Unlike BOMs or other complex structures, it is not likely that this query will have a performance impact.

However, keep in mind that once this limit is increased no other queries will be constrained and therefore other queries could possibly have a performance impact. It is important to carefully assess the potential performance impact.

Try It ... Increase Depth Limit of Query

1. Navigate to Administration->Variables
2. Create a new Variable ‘ac_QueryDepthLimit’
3. Assign a value of 3 and click Done
4. Return to the ‘**Document Project Programs**’ Derived Attribute to Save it successfully

Implementing the MAC Policy

- Derive all Programs where the User is a Member
- Derive all Program Parents of Projects Using Document
- Use a Boolean Expression to Compare Collections for Overlap (Intersection)**

```

Collection.Overlaps(
    CurrentItem.[Document Project Programs],
    CurrentUser.[User Programs]
)

```

Aras

27 | © 2023 Aras

Implementing the Use Case

As mentioned earlier, the business use case is to restrict access to Documents under Programs to those assigned as Members on these same Programs. Once both Derived Attributes are defined, we may build a MAC Policy that uses them to enforce this use case.

Try It ... Create and Activate the MAC Policy Using Derived Attributes

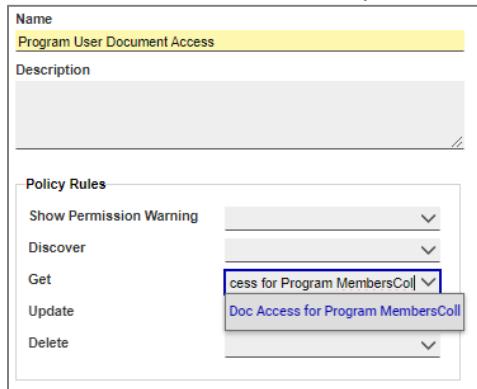
1. Navigate to Administration->Access Control->MAC Policies
2. Create a new MAC Policy, give it a name like '**Program User Document Access**'
3. Applied to Itemtype is '**Document**'
4. Save and use the sidebar menu to open the Rule Expression Editor
5. Add a new Rule giving it a name like '**Doc Access for Program Members**'
6. Use the following expression to define the Rule (note intellisense):

`Collection.Overlaps(CurrentItem.[Document Project Programs], CurrentUser.[User Programs])`

(this green checkmark indicates proper syntax, and can also be clicked to save expression)

Defining User Access With MAC Policies

7. Return to the Main MAC Policy Form and assign this Rule to the Access Right 'Get'



8. Click **Done** to Save the MAC Policy
9. From the More Menu [...] **Activate** the MAC Policy

The screenshot shows the 'Program Users Document Access' MAC policy details. It displays a table with one rule entry:

Name	Condition	Description
Doc Access for Program Members	Collection.Overlaps(SelectedItem.[Document Project Programs], CurrentUser.[User Programs])	

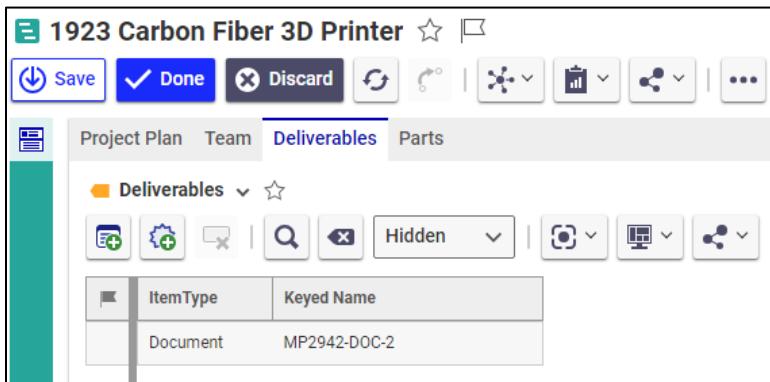
Below the table, a detailed view of the rule shows its condition: `Collection.Overlaps(SelectedItem.[Document Project Programs], CurrentUser.[User Programs])`. A tooltip for this condition is displayed, showing the full code: `1 Collection.Overlaps(SelectedItem.[Document Project Programs], CurrentUser.[User Programs])`.

Testing the MAC Policy

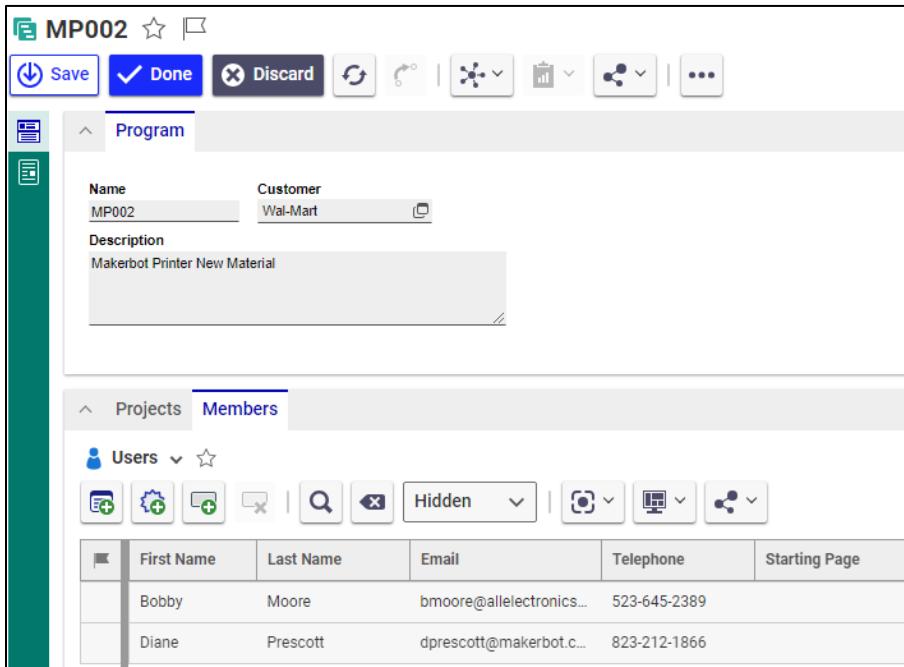
Once that the MAC Policy is active, Documents that are used as Deliverables on Project (child items) on any Program will only be accessible by Members of that Program. Let's try this out:

Try It ... Exercise the MAC Policy

1. Navigate to Portfolio->Programs in the TOC and open **MP002** for editing.
2. Open the '**1923 Carbon Fiber 3D Printer**' Project from the Projects relationship tab. Edit and add one or more Documents to the Deliverables tab, and click **Done**.

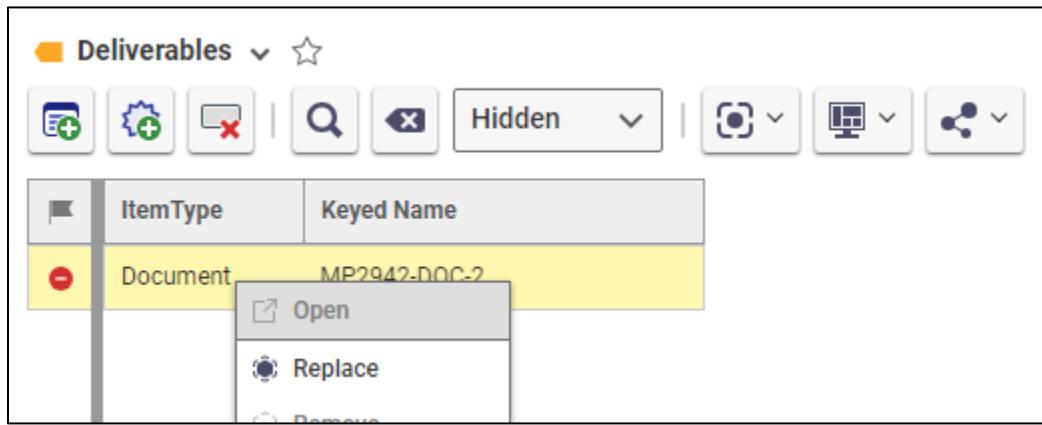


3. Return to Program MP002. Add some easily recognizable Users to the Member tab we created in an earlier exercise. Do not add Innovator Admin yet. BTW: If there is no Program User relationship on Program Itemtype, add it – then return to complete this step.



4. Save the Program.

- Remaining logged in as Admin, attempt to open 'Deliverables' (related Documents) on any Project child items to the Program.



Because we have associated 'Get' Access Right with this Rule, only those who are listed as Members in the parent Program will be able to open a Document Deliverable anywhere in this Program scope.

- From an incognito window, log in as **dprescott**, **bmoore**, or any of the Users you added to the Members tab on Program **MP002**.
- From Portfolio->Projects in the TOC open 1923 Carbon Fiber 3D Printer
- Open any Document from the 'Deliverables' tab. Close the Project window and return to the parent Program **MP002**.
- Return to the admin session window and add yourself (admin) to the Members list on Program **MP002**.
- Re-open 1923 Carbon Fiber 3D Printer Project and attempt to open the Document Deliverable.

Other Things to Try

- Add the Rule to different Access Rights, for instance 'Can Discover'
- Instead of adding to Members, add yourself to 'Exempt Identities' on the MAC Policy Itself
- Create more than one Rule and use various criteria on different Access Rights
- Define a new Rule to use a Property 'Security Level' on the Program such that:
 - Any Program Members with [Security Level] equal to or greater than that on the Project->Document Relationship Item can retain the Access Right(s) configured in a MAC Policy
- Explore other Use Cases...