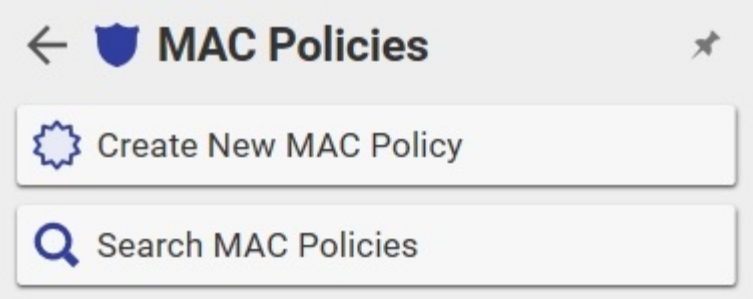
Establishing MAC Policies

Aras Innovator MAC Policies are used to control user access to Items through a set of MAC Policy Rules. MAC Policy Rules control access rights for Get, Update, Delete, Can Discover, and Show Permission Warning to the set of ItemTypes which the MAC Policy is applied to.

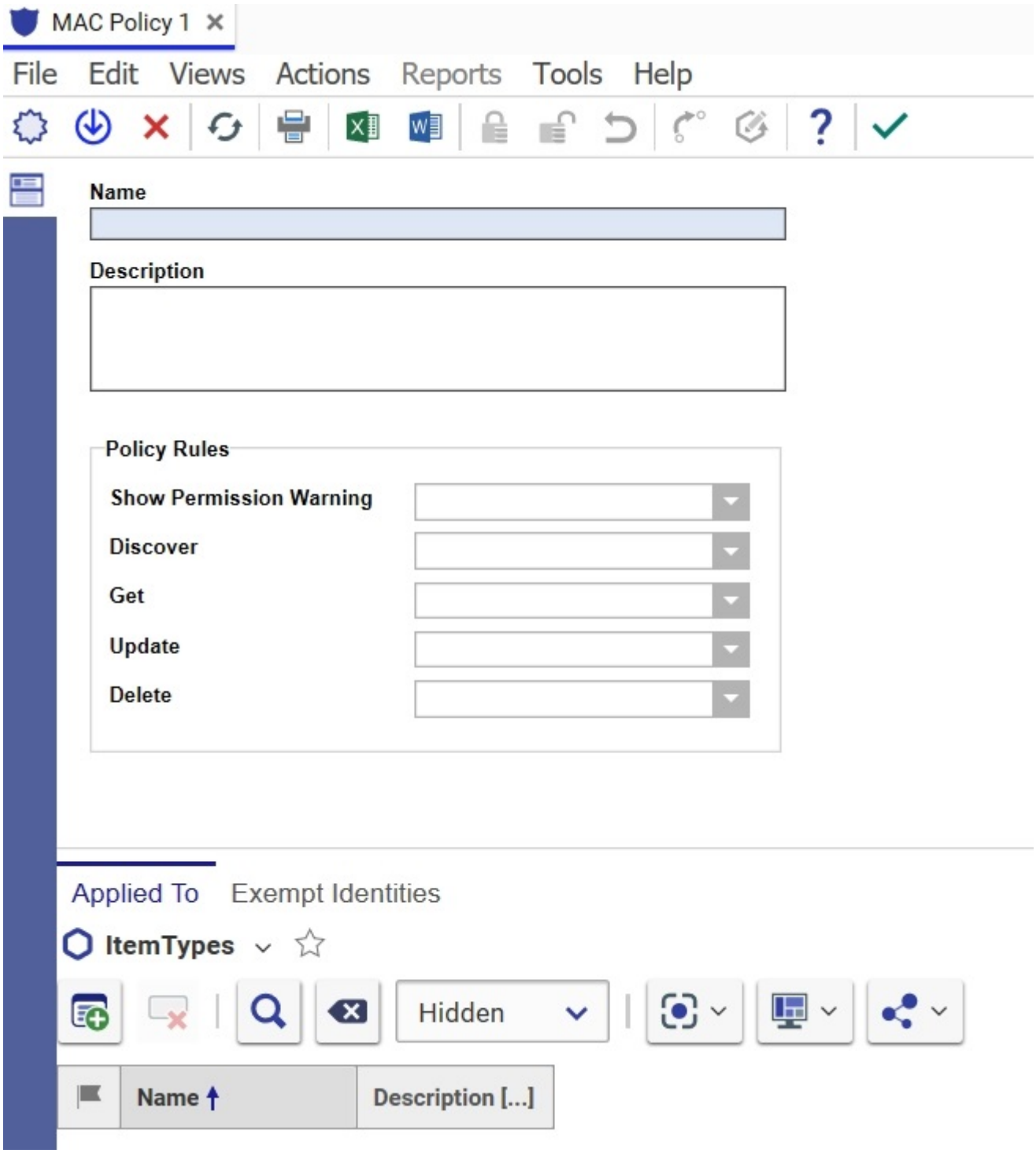
Creating a New MAC Policy

You can find MAC Policies in Aras Innovator by clicking **Administration** --> **Access Control** --> **MAC Policies** in the TOC. The following menu appears:

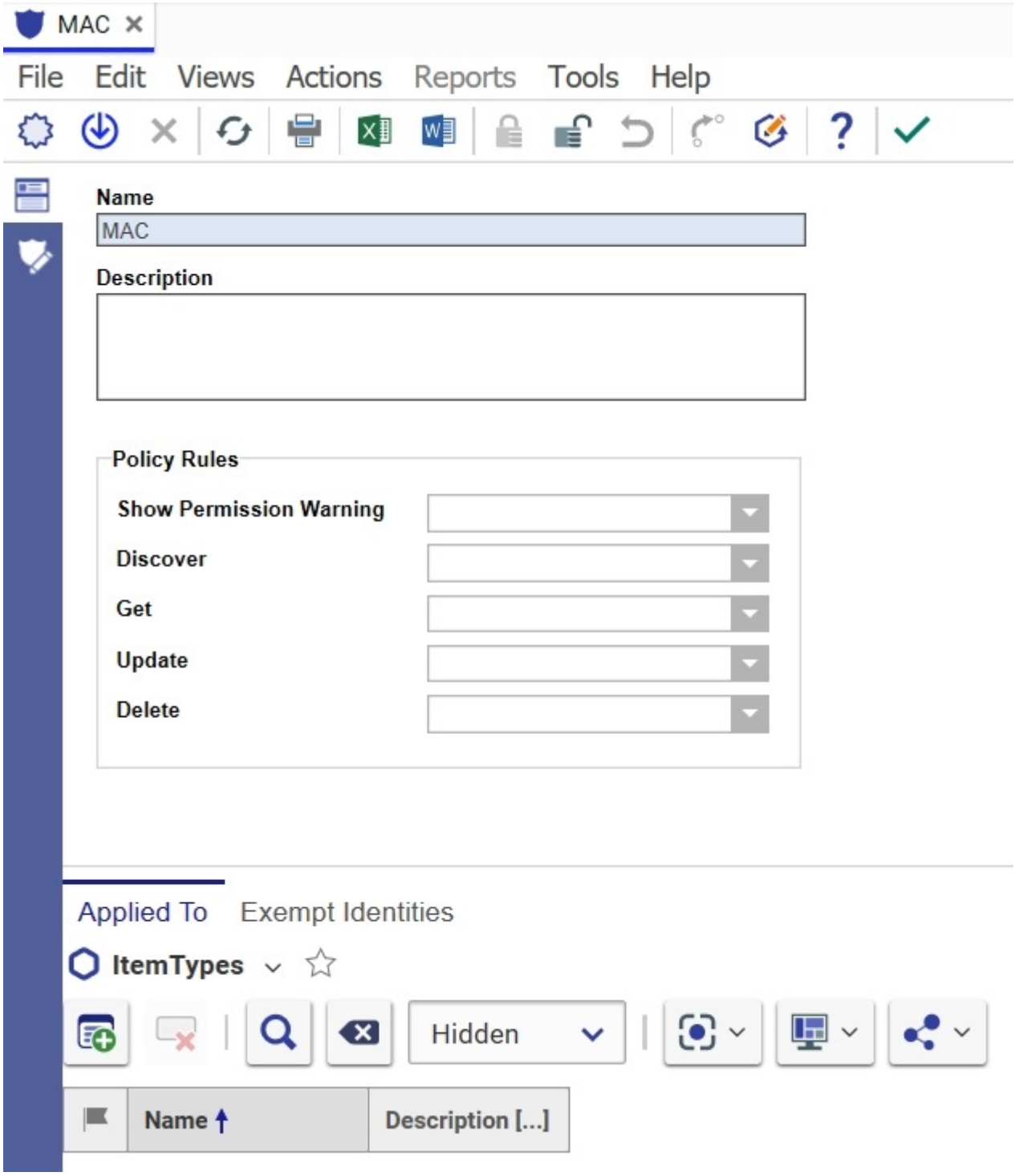


Only users with Administrative permissions have the ability to create MAC Policies. Once you create a MAC Policy, you must specify the Name and save it. Once you do that, you can create MAC Policy rules. Use the following procedure:

1. Click **Create New MAC Policy**. The following screen appears:
    1.



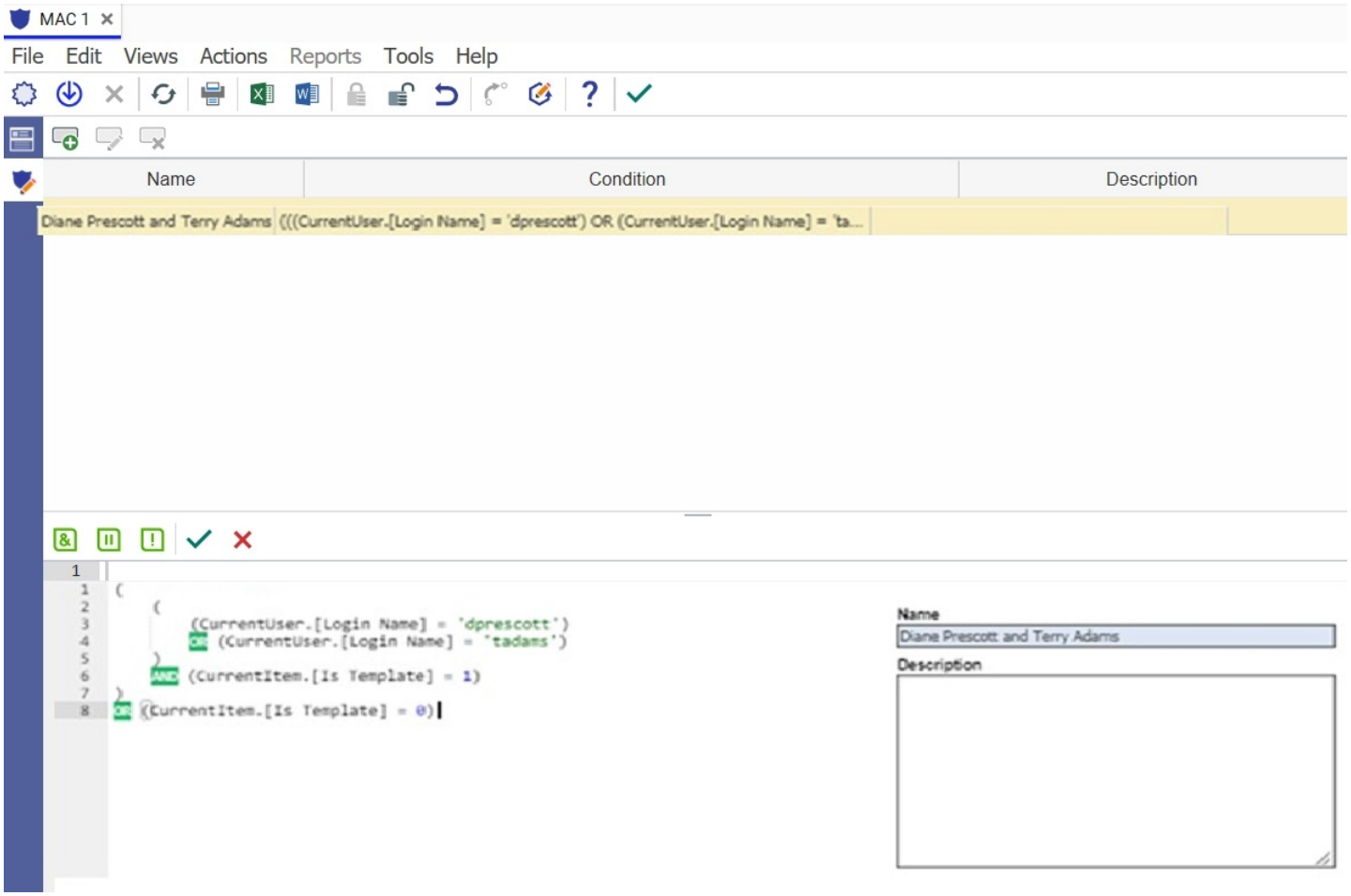
2. Enter the Policy Name in the Name field and click null . The MAC Policy Editor icon appears in the left margin.



Creating a MAC Condition

null To create a new Condition, switch to the MAC Policy Editor view by clicking on the MAC Policy Editor icon.

Selecting the New Condition icon null makes the lower portion of the window available to enter the Condition statement for the Policy Rule:



The access administrator can create N Conditions within any given policy. The Condition editing is done in the lower pane, and the upper frame displays saved Conditions previously created within this MAC Policy Item.

A MAC Condition is comprised of one or more Boolean expressions combined by Logical Operators AND, OR, and NOT. Boolean expressions can reference CurrentItem, CurrentUser attributes which may be:

- Property-based attribute obtained from the Item being accessed—CurrentItem. `<attribute_name>` .
- Property-based attribute from the User making access request—CurrentUser. `<attribute_name>` .
- Environment Attribute—dynamically generated on invocation via Method execution.
- Derived Multivalued Attribute—created using the Derived Attribute Definition item.
- xClass or xProperty reference used as an attribute.

Expressions may also use constant values (hard-coded values).

The following syntax rules apply to Policy Rule Condition statements:

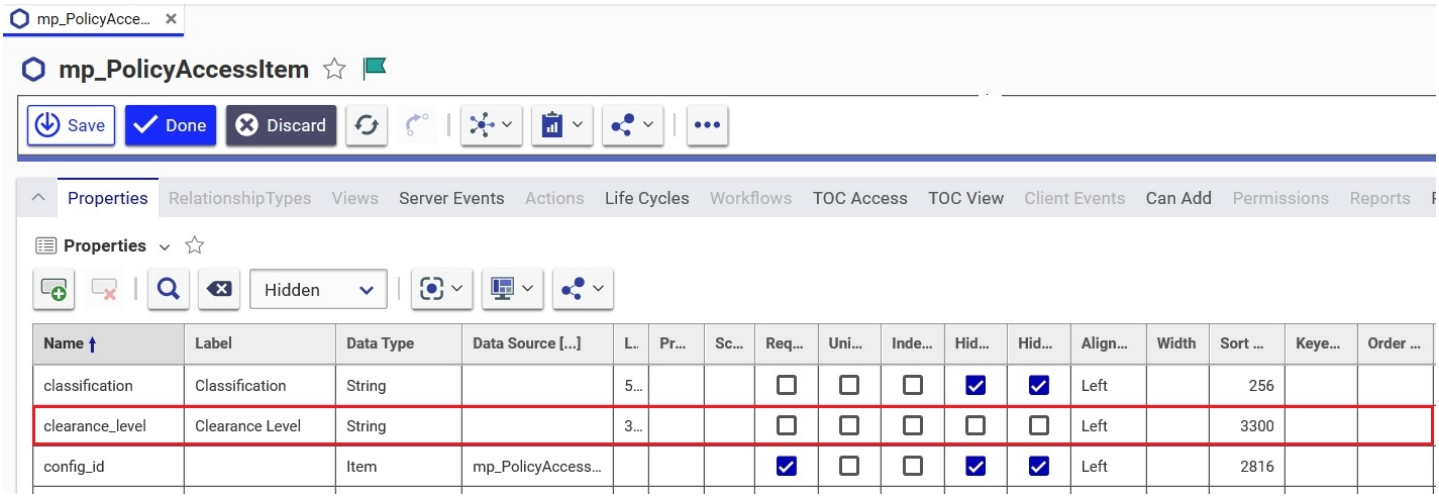
- Values are case sensitive.
- String literals text must be enclosed between quotation marks ('text'), quotation escapes with back slash (\').
- A Constant can act as an operand when you use it in a comparison. Otherwise, a Constant is a string type.
- Operators and precedence are the same as those used in SQL languages.
- Operators are not case sensitive.
- Arithmetic operators are not supported.
- Parentheses can be used to override operator precedence.
- Comparing two strings follows Transact-SQL rules.

Important

If the Policy Rule uses a Property on the User ItemType and a user can edit their own User Item, then the user is capable of changing their level of access to any of the ItemTypes that a MAC Policy is applied to. This can also apply if the Edit control is not restricted in a MAC Policy and other Permissions are based on editable Properties.

Adding Supported ItemType Properties

Only properties that are attached to the **mp_PolicyAccessItem** ItemType can be referenced in a Policy Rule Condition statement. To add additional properties, open the **PolicyAccessItem** ItemType and add properties that will need to be referenced by the MAC Policy Rules.



Properties added to the PolicyAccessItem must exactly match the properties of the ItemTypes that the MAC Policies are being applied to. All ItemTypes that the MAC Policy is being applied to must have the property, which is being referenced, otherwise validation will fail when the Admin attempts to save the Policy. The following data types are supported:

- String
- Integer
- Float
- Decimal
- Boolean
- Date
- Item
- List

In the previous example, the **clearance_level** property has been added and thus a Policy Rule can reference the property using the following syntax: **CurrentItem.[Clearance Level]**.

Supported Comparison Operators for Boolean Expressions

Table 4 lists operators that you can in the Condition statement of a Policy Rule.

1. Supported comparison operators for Boolean expressions

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.

Available Helper Methods

Table 5 lists available methods that you can use within the Condition statement of a Policy Rule.

Table 1: Available helper methods

Method	Response
CurrentUser.IsMemberOf(`<Identity Name>`)	Returns true if the current user is a member of a (non-system) Identity `<Identity Name>` , otherwise it returns false.
CurrentUser.IsMemberOf(Property `<Item>`)	Returns true if the current user is a member of the Item Property of type Identity. For example: CurrentUser.IsMemberOf(CurrentItem.identity_id)
CurrentUser.IsMemberOf(`<multival attribute>`)	Returns true if the current user is a member of at least one of (non-system) Identities from Collection `<multival attribute>`
String.Contains(`<StringToSearch>` , `<SearchForString>`)	Returns true if `<SearchForString>` is a substring of `<StringToSearch>` , otherwise it returns false.
CurrentItem.HasUserVisibilityPolicyAccess()	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.

Using xClasses and xProperties in MAC Policy Conditions

You can use xProperties that are associated with the mp_PolicyAccessItem ItemType as item attributes in MAC Policy conditions. You should specify supported xProperties in the Allowed

xProperties relationship tab in the mp_PolicyAccessItem. You can also use xProperties that are associated with the User ItemType as user attributes. Using xProperties in MAC Policy conditions may cause some performance penalties but it has the advantage of being able to specify access control for an item xProperty independently from access control for the item itself (and therefore all the item regular properties).

The rules for calculating a MAC Policy condition that is using an xProperty in cases when the xProperty is undefined follow those for AML. You can explicitly check in a MAC Policy condition to see if an xProperty is defined on an item or a user using the built-in functions described in section 2.1.2.1 .

Important

DO NOT edit xProperty definitions that are being used in active MAC policies. Doing so results in a system failure. To adjust xProperty definitions, you must first disable the MAC policy. When you finish updating the xProperty definition, re-enable the MAC policy.

In MAC Policy conditions you can also check if an item or a user is classified by an xClass using the built-in functions described in section 2.1.2.1 .

Using Methods to Verify Item Classification

Table 6 lists methods that enable you to check in MAC policy conditions to see if an item or user is classified using a particular xClass or xProperty.

Table 4: Methods for Item classification verification

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

For more information about xClasses and xProperties, refer to the Extended Classification guide.

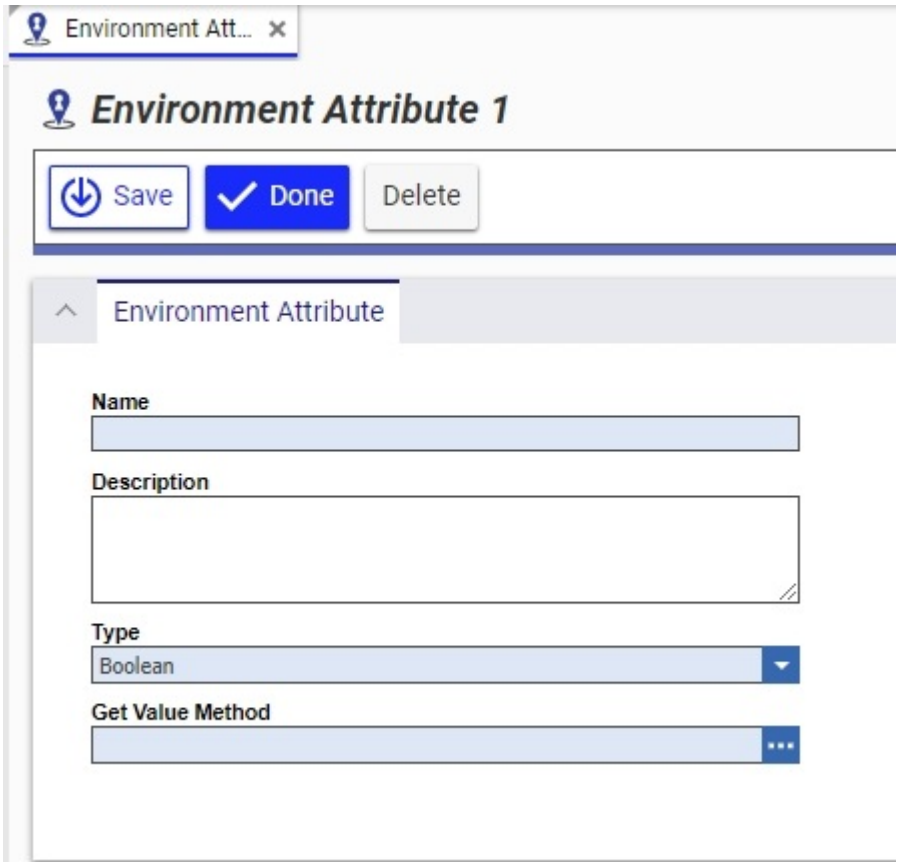
Defining Environment Attributes

Environment attributes grant a user certain access rights to an Item based on specific circumstances such as the geographical location or the time an access request is made. For example, if the user makes the request outside of work hours, the request is denied. If the same user makes the same request during work hours, their request is accepted.

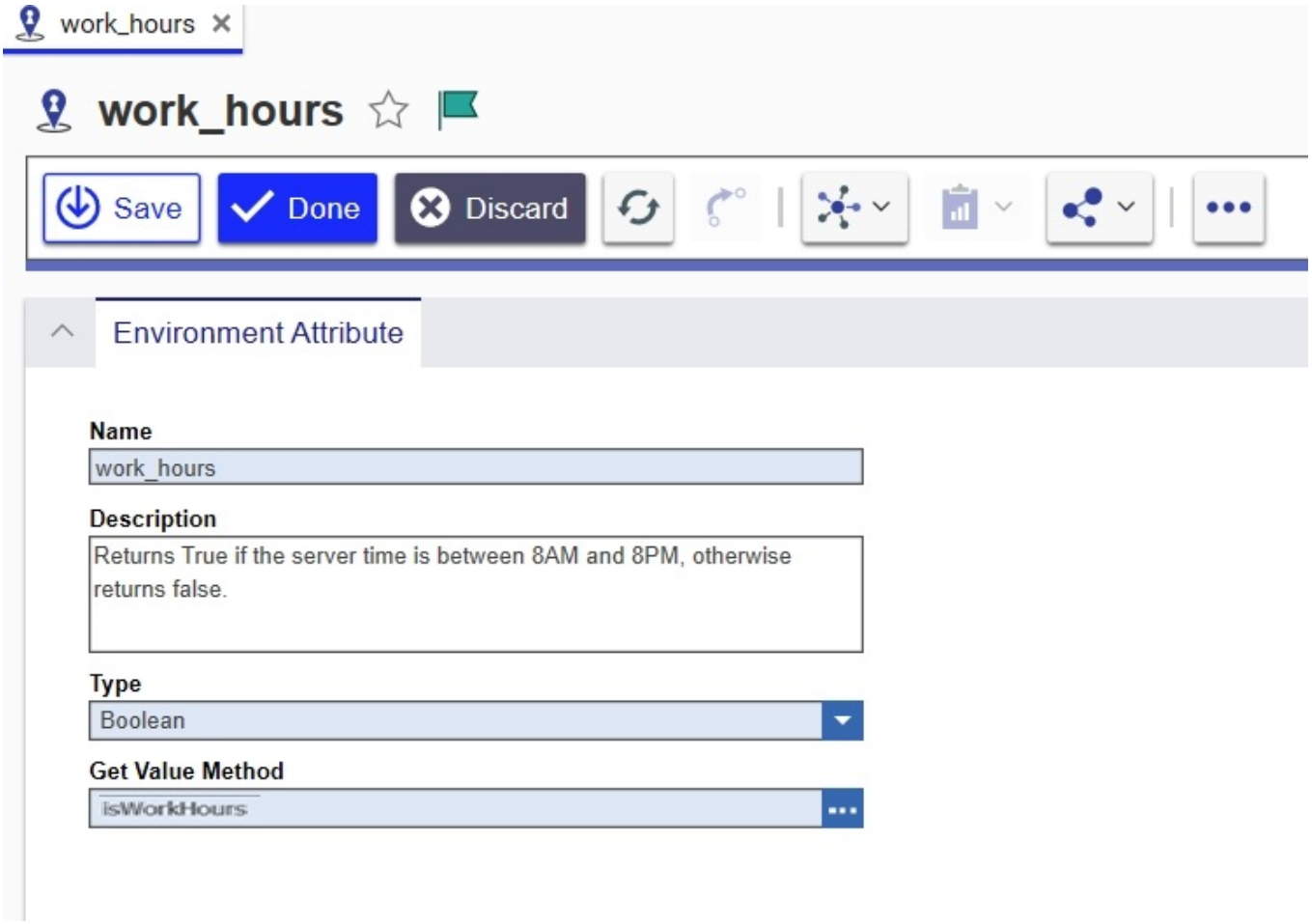
To create Environment Attributes in Aras Innovator:

1. Click **Administration** --> **Access Control** --> **Environment Attributes** in the TOC. The menu shown in figure 16 appears.

2. Click **Create New Environment Attribute**. A blank **EnvironmentAttribute** dialog appears.



3. Enter the appropriate information in the **Name**, **Description**, **Type**, and **Get Value Method** fields.



4. Click null to save and unclaim the attribute.

Each attribute has a unique Name, Type, and custom method (specified in the “Get Value Method” property) that is called by Aras Innovator to get the attribute value for each request to Aras Innovator. The supported data types for Environment Attributes are Boolean, String, and Integer. The environment attribute values are considered when calculating MAC Policy conditions for determining if an access request should be granted if the conditions use the environment security attribute. An environment security attribute with the name `<attr_name>` is referenced in a MAC Condition as ‘$ `<attr_name>` ’.

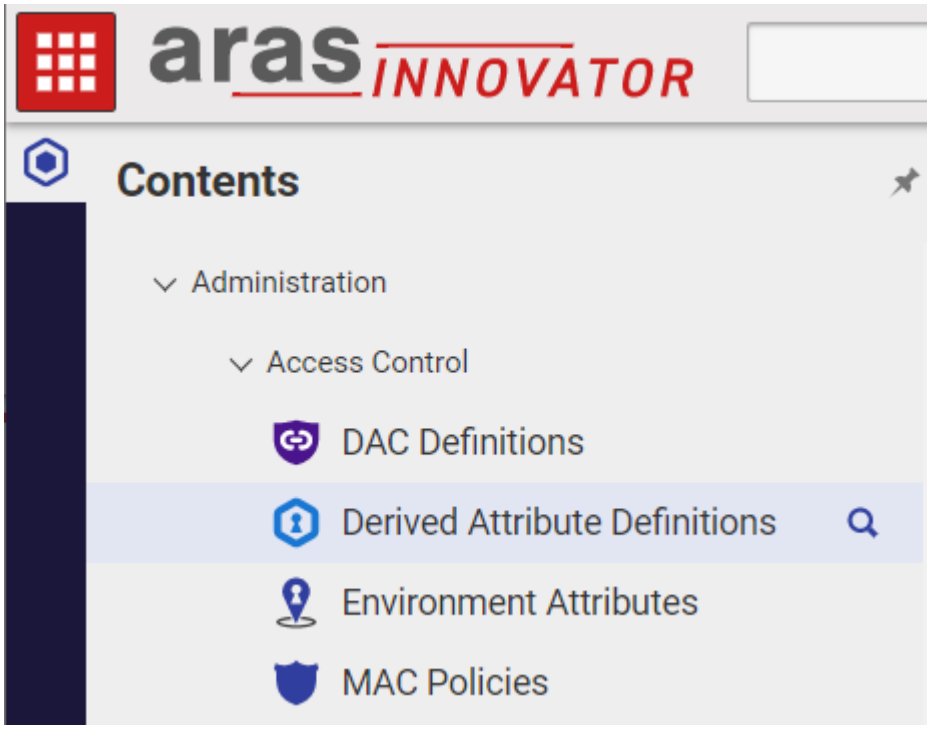
See Section 3.1 for an example of a method that can be specified in the Get Value Method field.

Important

It is the responsibility of the system administrator/implementation specialist to create custom server methods that return the environment security attribute non-NULL values to Aras Innovator. The custom methods should be created using the EnvironmentAttributeMethod method template.

Derived (Multi-Valued) Attributes

These special attributes are configured independently (like Environment Attributes) and become available in the Condition Editor (Section 2.1.1) when successfully configured. Derived Attribute Definitions reside under **Administration** --> **Access Control** in the TOC.



Derived Attributes use an embedded Query Definition to specify the Path from the Root (CurrentItem) to a related Leaf Item and its Target Property from which the Derived Attribute values are extracted.

Refer to the following section for an explanation of Derived Attributes.

Configuring a Derived Attribute

A Derived Attribute has the structure shown in Figure 20 and described in Table.

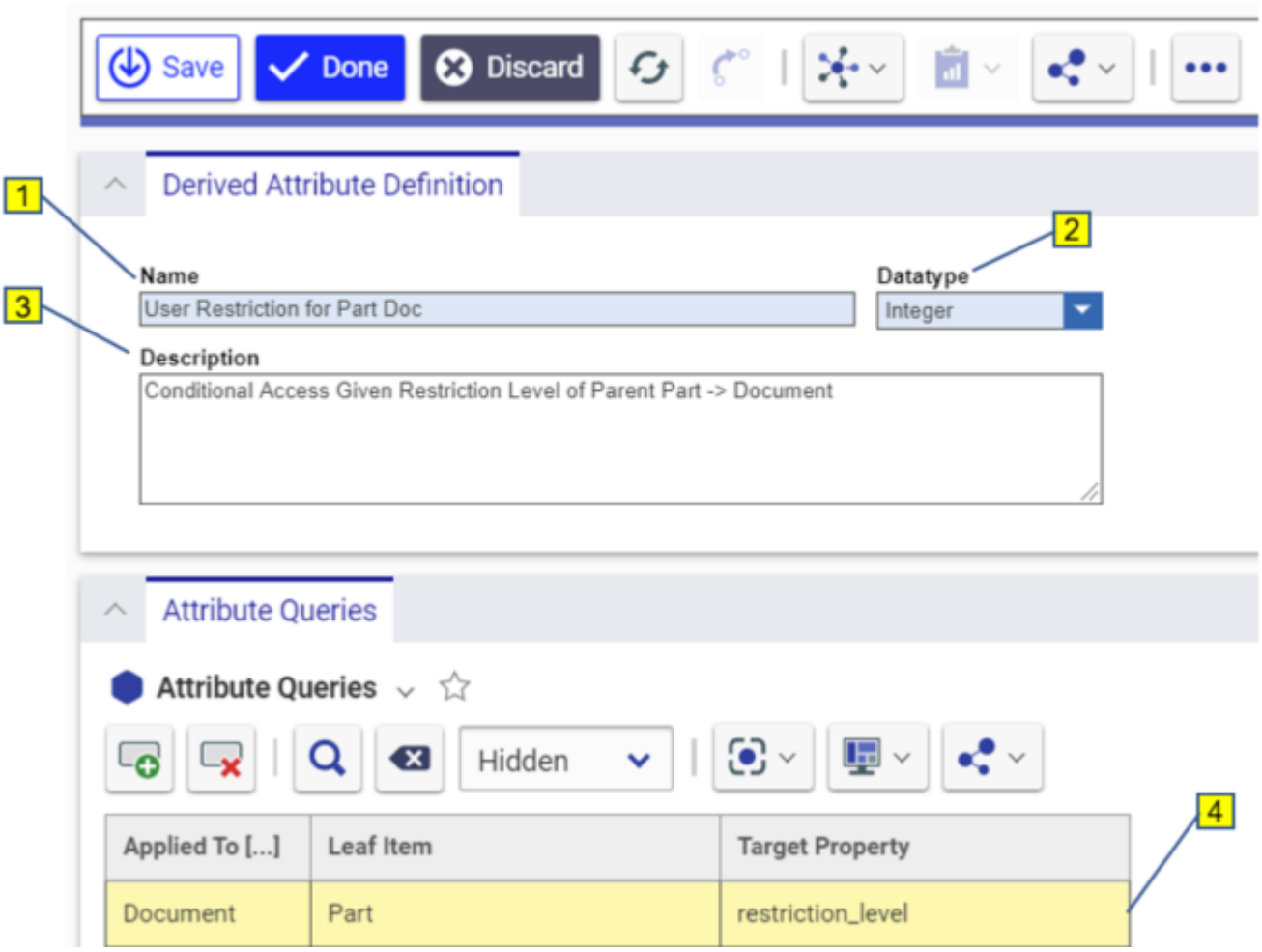
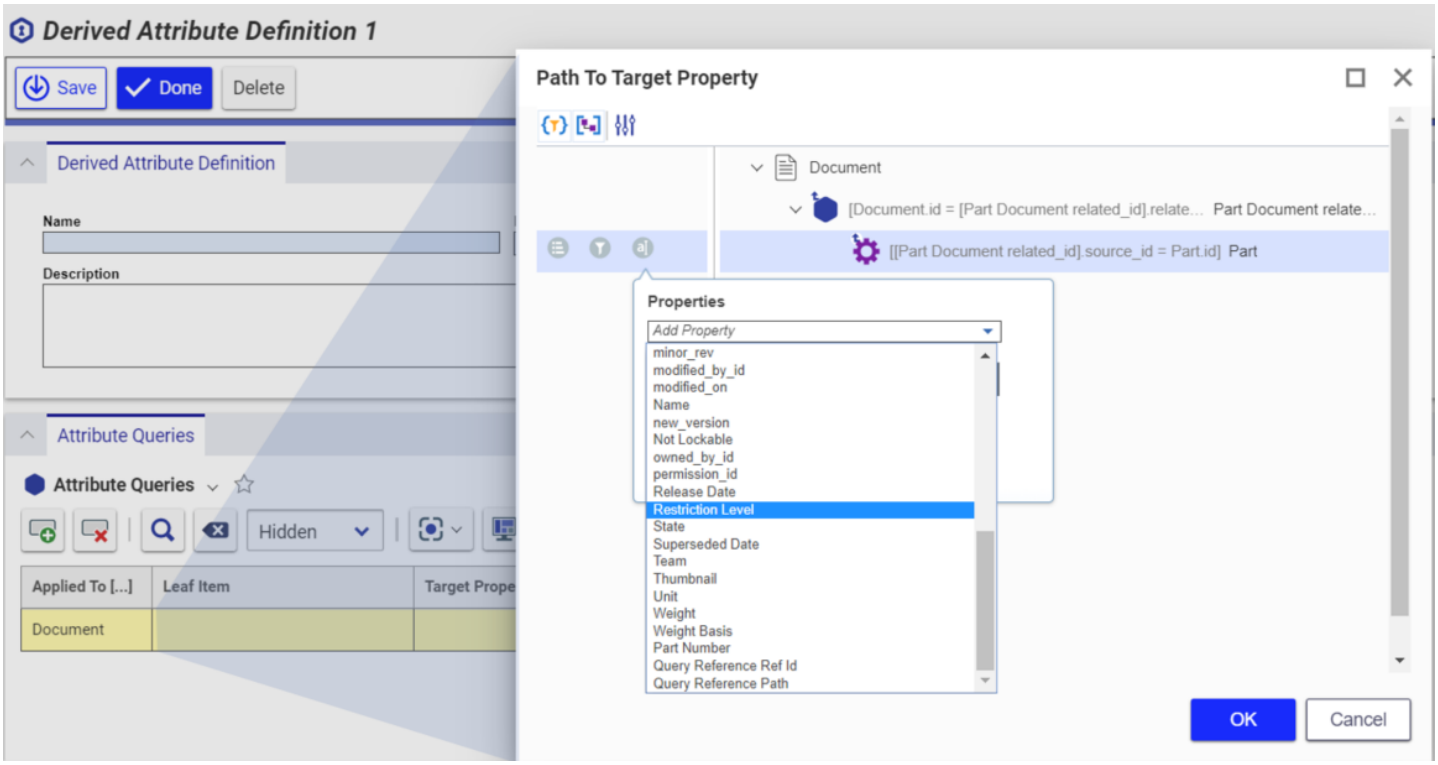


Table 5: The Derived Attribute structure

1	**Name**	**Symbol by which this derived attribute is referenced (Required). Enclose spaces in [square brackets] when used in an expression.**
2	**Datatype**	Datatype of values contained in a derived attribute collection (Required). Must match Target Property type, and vice-versa.
3	**Description**	Text description of Attribute for reference.
4	**Attribute Query**	A key Relationshiptype that defines how to derive values for this attribute:
First-time Entry Only	**=> "Applied To" Itemtype**	The Itemtype that will define the scope of "CurrentItem" in Boolean expressions; also, the Root Item of the Path defined by the *Embedded Query Definition*. This field is a Required initial entry and becomes read-only once the Path is defined.

Embedded (Not visible)	=> **(Query Definition)**	Internally stored query definition describing (a) the Path from Root item to Leaf item and (b) Target Property on the Leaf item to derive into the collection (see diagram below).
R/O	=> **Leaf Item**	The Item at the endpoint of the Query Path. Target Properties are derived from this item. There can only be one Leaf Item in the query path.
R/O	=> **Target Property**	Name of the Property on the Leaf Item used to derive Attribute values. Type must match Datatype [2].

The **Query Definition** is accessed by *double-clicking the **Leaf Item** column** of the Attribute Query. The Root item is pre-populated as the **Applied To** Itemtype. Using the standard QD user interface, define the Path from Root to Leaf Item, and also select the Target Property. On completion, the read-only **Leaf Item** and **Target Property** columns will be populated.



Using a derived attribute in a Boolean expression

Derived Multivalued Attributes are Collections, and the operators described in Table 8 support their use in Condition logic (Boolean expressions).

Table 6: Collection operators

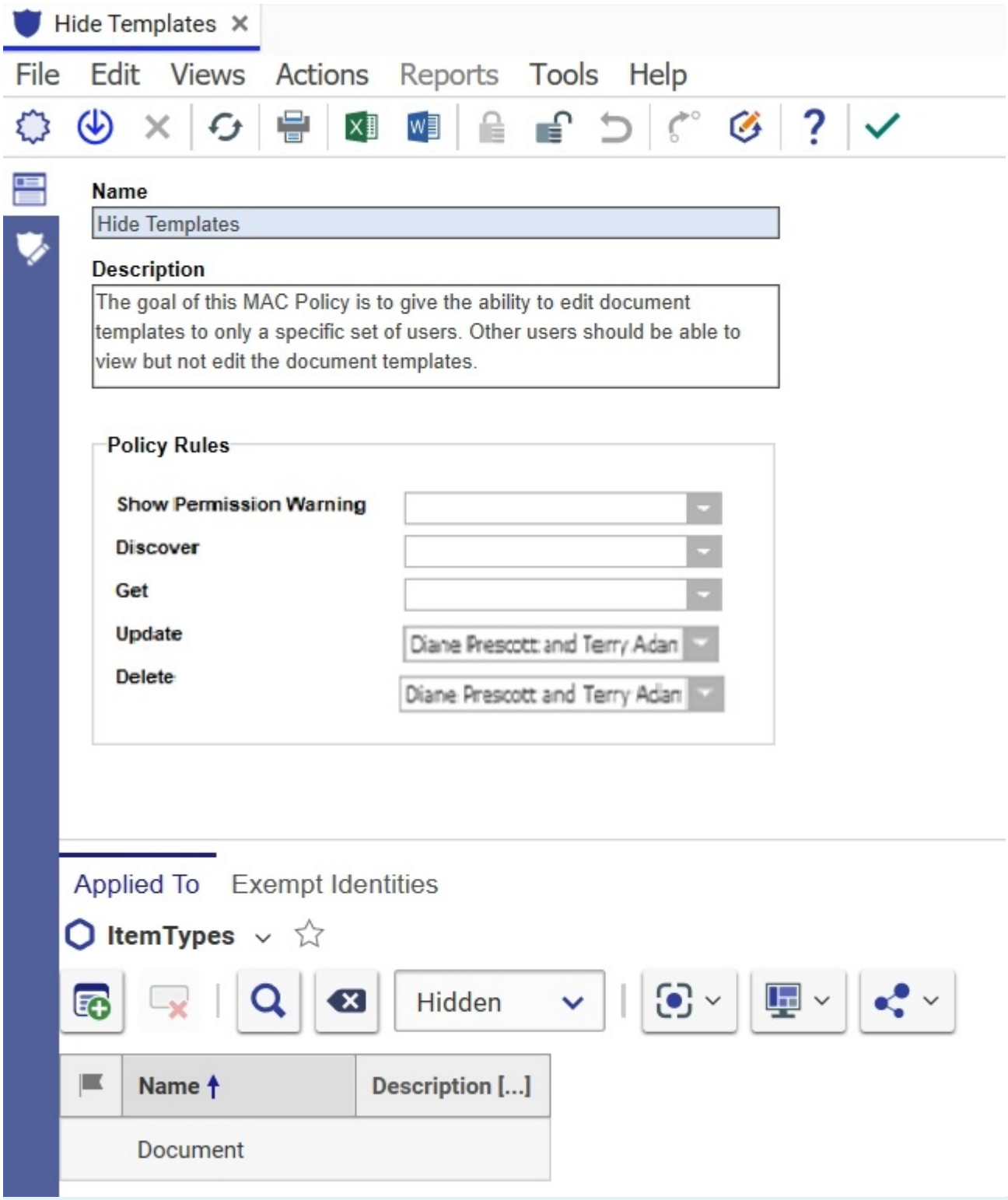
Operator	Description
`Collection.IsEmpty(<multival attribute>)`	TRUE if Collection is empty
`Collection.Contains(<multival attribute>, value)`	TRUE if Collection `<multival attribute>` contains the value `<value>` (type
`Collection.Overlaps(<multival attribute1>, <multival attribute2>)`	TRUE if Collection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

Applying Policy Rules

Once you create Conditions, you may assign them to specific Access Rights. There are five drop down lists:

- **Show Permission Warning** – determines if, when permissions are restricted, the user receives a standard permission warning.
- **Discover** – determines if the user can view an Item within a search grid.
- **Get** – determines if the user can open Forms and view all the Properties of an Item.
- **Update** – determines if a user can claim an Item, make changes, and save those changes.
- **Delete** – determines if a user can delete an Item.

Selecting a Condition from the dropdown list applies that Condition to the access right. The user is denied access if the selected conditions are not met.



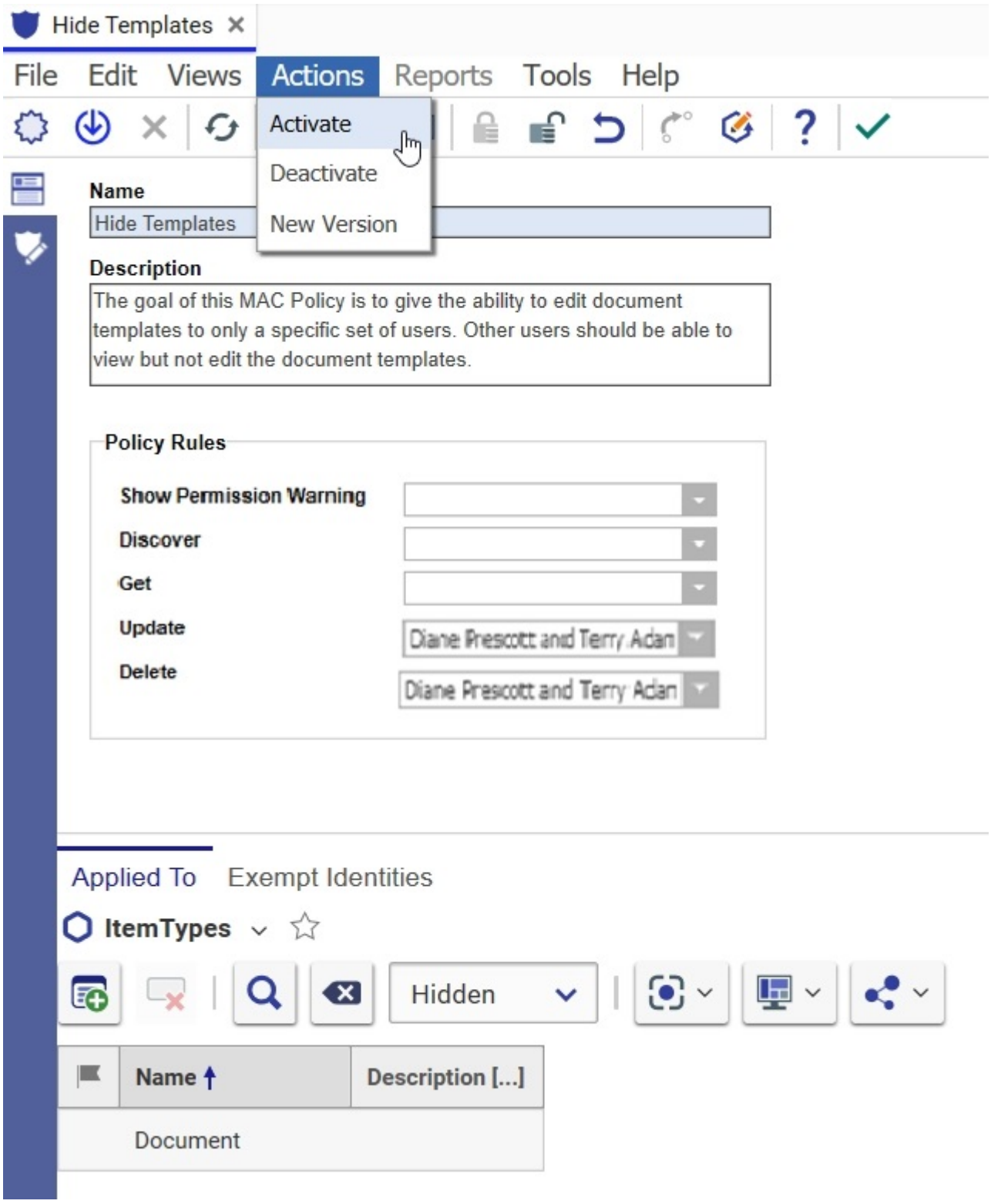
Important

If a Rule is not selected for an access right, then the MAC Policy will not apply any restrictions to that access right.Standard Aras Innovator role-based permissions and all applicable MAC Policy rules must grant access to an Item before a user is able to access it.

Applying MAC Policies

Once you establish the Policy Rules, you must assign the MAC Policy to the desired ItemTypes and then activate the MAC policy. Assigning the MAC policy to ItemTypes is done by adding them to the Applied To Relationship.

To activate a MAC Policy, unclaim the policy and select the ‘Activate’ Action. When Activating a MAC Policy, all users except for the admin activating the MAC Policy should be logged off and prevented from accessing the system until the MAC Policy is activated.



Updating MAC Policies

Once activated, a MAC Policy can be deactivated or versioned. The **Deactivate** action sets a MAC Policy to the **Inactive** state. When a MAC Policy is **Inactive**, the Policy Rules are not applied to control access.

To modify a previously active MAC Policy, the Policy must be versioned through the **New Version** Action. The previous revision of the MAC Policy is promoted to the **Archived** state only after the new version of the Policy is activated.

Exempt Identities

Each MAC Policy includes a list of “Exempt identities.” If the user making an access request has an identity that is included in the “Exempt identities” list, MAC Policy rules are not applicable to the request.