

Property Management Rules & Rules definition file

Version du document : 2
Version MySurvey : 1.7.7.6

Date : 01/06/2022

Introduction: Property Management Rules

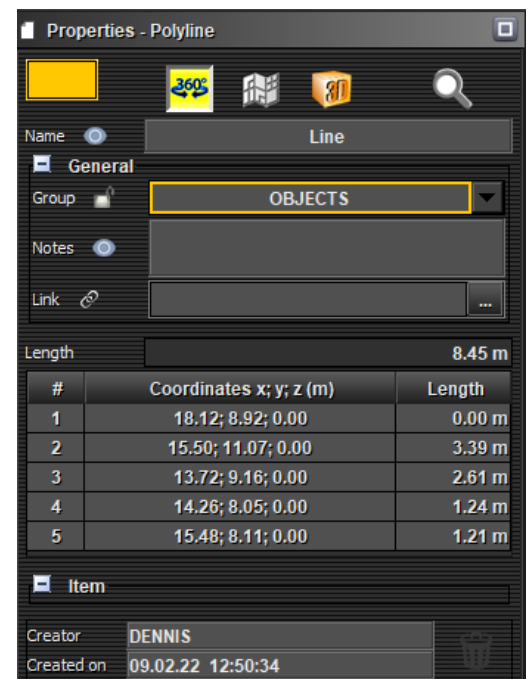
Items and Groups in MySurvey are considered “Property Providers” because they store a defined list of *properties* internally.

A property itself is defined by a *key/value* pair such as NAME → “Box14”.

The available properties and their access (read/write) depend upon the type of provider.

The Property Manager (PM) is the panel window in MySurvey that displays all the currently selected provider’s properties.

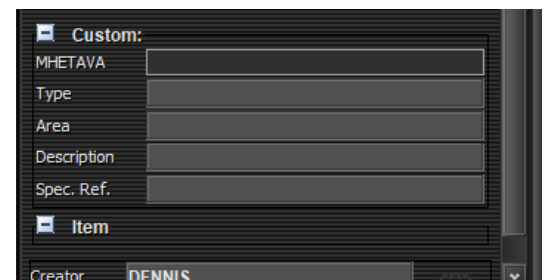
This may include things like the name, parent group, comments, links to attachments, geometric attributes, creator, date, etc.



Currently, the only portion of panel that is customizable is the section labeled “Custom”.

Here, attributes will be placed when the conditions are met and using a scripting language that will be described below.

Users will be able to enter String values into the fields and they will be stored in the database.



Property Management Rules & Rules definition file

Rules Script

To manage properties in a data-driven manner, the *main.rules* script file may contain a sophisticated set of instructions (conditions and actions) defining custom behavior in MySurvey.

Custom Properties (Attributes)

Additional custom properties can now also be attached as metadata (also referred to as “attributes”) in MySurvey. The user may define a string *key* of their choosing (must not conflict with the built-in keys listed below) and can then attach a string *value* to it as metadata.

An attribute structure’s syntax begins with the *new* keyword, closes with the *end* keyword, and may contain a list of optional parameters written as <key>=<value>.

Example Attribute Structure for adding a single custom property:

```
new Attribute "TEMPERATURE"
  CAPTION="Temp. °C"
  TOOLTIP="The highest daily internal temperature measured in Celsius"
  TYPE_ATTRIBUT=STRING
  EDIT=ENABLED
  BUBBLE=COLLAPSED
end
```

Below is a description of each parameter, some of which have multiple valid keys in English and French:

Key	Parameter Description
CAPTION	This is the custom property’s name that will be displayed in the PM field label, CSV column header, etc. It is case-insensitive!
TOOLTIP	This is an optional informative text (sentence) to show when the mouse cursor hovers over the PM field corresponding to this custom property.
TYPE_ATTRIBUT	The type of variable storing the data, for now only <i>STRING</i> is supported.
EDIT	Whether the value can be edited in the PM field. Possible values: <i>READ_ONLY/ENABLED</i>
BUBBLE	How to display the custom property in the popup bubble: <i>SHOW/HIDE/COLLAPSED</i>

Importing custom properties (attributes)

When loading an RVM file, an optional ATT file with the same name may also be loaded to obtain and merge extra metadata to the property providers, even if the RVM is a subtree of the ATT tree.

Property Management Rules & Rules definition file

Script Syntax

The script syntax is similar to the C language style, but with simplifications for improved readability.

<code>//Single line comment</code>	<i>All text until the end of the line is ignored by the parser.</i>
<code>/* Multiline comment */</code>	<i>All text between the begin and end tokens is ignored by the parser.</i>
<code>"String literal" "with \"escape\" chars"</code>	<i>String literals are surrounded by double quotes and may contain spaces and escape characters using the backslash \. These are mandatory for custom names, paths, etc. to avoid conflicts with keywords.</i>
<code>["el1","el2","elN"]</code>	<i>Arrays are a grouping of multiple elements bounded by square brackets with comma delimiters.</i>
<code>["root"/"parent"/"child"]</code>	<i>Paths are like arrays but are delimited by forward slashes /</i>
<code>methodName("arg1","arg2","argN")</code>	<i>Methods have an optional argument list that are like arrays but bounded by parenthesis.</i>

Below is a list of the reserved keywords and their meaning.

Reserved keywords	
<code>if, else, end</code>	<i>Conditional branch structure keywords. See examples below.</i>
<code>and, or, not</code>	<i>Conditional linkers. See examples below.</i>
<code>equals, in, contains</code>	<i>Binary comparison operators with Boolean result.</i>
<code>for, break, continue, end</code>	<i>Iterative structure keywords (Not yet available).</i>
<code>return, exit</code>	<i>Reserved (Not yet available)</i>
<code>alias</code>	<i>Usage is: <code>alias dest source</code>. Does a simple find/replace of all instances matching the word <code>dest</code> with <code>source</code>. A warning will be given if a duplicate alias is defined, but it is legal and will overwrite the previous definition with the new one. The run-time result always uses the last defined alias.</i>
<code>import</code>	<i>Scans the specified file before proceeding to the next line, like how C <code>#include</code> works.</i>

Built-in methods	
<code>prop(type)</code>	<i>Retrieves the property with the given identifier. This identifier can be any custom attribute's TYPE or one of the many built-in property identifiers. See Property list.</i>
<code>path(type)</code>	<i>Retrieves the ancestor path (including self) of the requested property. If no property is specified the default NAME property is used.</i>
<code>showInBubble(type, override)</code>	<i>Display a new custom property in the Tooltip Bubble (text line)</i>
<code>setAsRemarkable()</code>	<i>Define a remarkable assembly item</i>
<code>assemble()</code>	<i>Define a group to be auto-assembled (during import)</i>
<code>addColumnCSV(type, override)</code>	<i>(Not yet available) Display a custom property in exported files (CSV column)</i>

Property Management Rules & Rules definition file

<code>addColumnAssembly(type)</code>	Display an additional column in the PM assembly list.
<code>addImportMeta</code>	Add a metadata key/value pair once during import.
<code>showInPM(prop)</code>	Display a property <i><prop></i> in the PM panel (text field).
<code>propAncestor(prop, PROP_ANCESTOR, VALUE_ANCESTOR_1, ..., VALUE_ANCESTOR_N)</code>	Retrieves the value of an ancestor's property <i><prop></i> . A suitable ancestor is determined by comparing a matching ancestor's <i><PROP_ANCESTOR></i> value to the list of allowable values: <i><VALUE_ANCESTOR_1>, ..., <VALUE_ANCESTOR_N></i>
<code>propAncestorIfUnset</code>	Same as <code>propAncestor</code> but retrieves "self" property first if available.
<code>setFlag(key, value)</code>	Add (or overwrite) a key/value pair of strings into the global variable list. A list of pre-defined keys and values are shown later.

Syntax Color Highlighting

For easier reading, a `rulesNpp.xml` file has been included for use with Notepad++ to highlight syntax colors of the keywords in any *.rules file.

Type safety

Very little checking is done at compile-time to ensure operands/arguments are of the proper types for their respective operators/methods. This may result in run-time errors, but they will be accompanied by detailed error reports with the line numbers where the issues occurred.

Examples

In this example, two aliases are used to make the rest of the code easier to read. Then depending on the provider's custom property "TYPE", it will either display the temperature value in the bubble tooltip or the PM.

```
alias type prop("TYPE") //Shorthand for obtaining a provider's type
alias T "TEMPERATURE" //Create a shorthand of the property name

if type in ["PIPE","HVAC","EQUI"] //If the provider's type is either PIPE, HVAC, or EQUI
    showInBubble(T) //Then show the temperature in the tooltip bubble
else
    showInPM(T) //Otherwise show the temperature in a PM field
end
```

Alias can be used to simplify an array representation. Conditions can also be joined by `and`, `or`, `else if`, etc.

```
alias SUBCOMPS ["BRAN","TUBI","DAMP","SILE","ACCES PANEL"] //shorthand array

if type in SUBCOMPS //If the provider's type is contained in the aliased array
or type in ["PIPE","HVAC","EQUI"] //Or If the provider's type is either PIPE, HVAC, or EQUI
    showInBubble(T) //Show the temperature in the tooltip bubble
else if type equals "ATTA" //Otherwise if the provider's type is ATTA
    assemble() //Assemble/pack during import
else
    showInPM(T) //Otherwise, add a temperature field in PM
end
```

Property Management Rules & Rules definition file

Overriding a property by using an ancestor's value:

```
propAncestorIfUnset("NAME", "TYPE", "BRAN", "PIPE", "HVAC")
```

Explanation: Returns the NAME value of the current provider if it has a NAME, otherwise it will search upward starting with its parent's property key "TYPE", which must be a value of either "BRAN", "PIPE", or "HVAC". If none of those ancestors are found, returns an empty result.

Using paths of ancestor types in complex conditions:

```
alias pathType path("TYPE") //Shorthand for obtaining a path consisting of ancestors' types

if type in ["PIPE","HVAC","EQUI"] //If the provider's type is pipe, HVAC, or equipment
  showInBubble(T) //Show the temperature in the tooltip bubble
//The direct grandparent and parent must be of type pipe and branch respectively
else if pathType equals ["**"/"PIPE"/"BRAN"/"?"]
  showInBubble(T,propAncestor(T,"TYPE","BRAN","PIPE"))
else if pathType contains ["EQUI"/"?"/"**"] //EQUI must be an ancestor (parent or higher)
  showInBubble(T)
else if pathType contains ["BRAN"] //Provider's type must be BRAN or have BRAN ancestor
and type in SUBCOMPS //Provider's type must be listed in the array
  showInBubble(T)
else if pathType equals ["**"/"STRU"/"?"] //Direct parent must be of type structure
  showInBubble(T)
//Provider type must be an Equipment with a parent Structure
else if pathType equals ["**"/"STRU"/"EQUI"]
  //Provider should inherit temperature from first available equipment or structure
  showInBubble(T,propAncestorIfUnset(T,"TYPE","EQUI","STRU"))
end
```

Users will have access to a limited set of built-in properties using the keys list in the table below. The exhaustive list (to be expanded) of properties in MySurvey:

Built-in Property Keys		
Key	Access	Description
Base		
NAME	RW	The provider's name.
TYPE_MS	R	The provider's (built-in) MySurvey type. See the full list TYPES_MS below
Cylinder		
DIAMETER	RW	Cylindrical Diameter
RADIUS	RW	Cylindrical Radius
HEIGHT	RW	Cylindrical Height
CENTER_BOTTOM	RW	The center of the cylinder's bottom circular face
CENTER_TOP	RW	The center of the cylinder's top circular face
AREAUV	R	The cylinder's rectangular are (diameter x height)
VOLUME	R	The cylinder's 3D volume
...		

Property Management Rules & Rules definition file

Global flags/variables

Calling `setFlag(key, value)` will add or overwrite the entry with the specified key (case-sensitive) paired with the corresponding string value. It can be conceptualized as static global variables used within the context of the rules where both the key and value are stored as strings that can be evaluated at run-time.

Pre-defined global variable flag keys and values

Below is a list of the available predefined keys and accepted values and their purpose. Many are "Booleans" which are expecting either "TRUE" or "FALSE" (typically anything other than "TRUE" is also considered false).

Key	Type	Purpose
"SHOW_EMPTY_BUBBLES"	Boolean	When "TRUE", custom properties which have empty string values are NOT shown in the bubble tooltip popup (even if requested by <code>showInBubble</code>).
"PULL_UP_ONLY_CHILD"	Boolean	When "TRUE", after importing a file of 3D models, a post-processing step is performed to recognize and pull-up any child item that is alone within its parent group.
"METADATA_KEY_AS_PROP"	Boolean	When "TRUE", all imported metadata keys will be interpreted as potential custom properties (attributes) and added automatically. They can ALL be shown via wildcard: <code>showInBubble("*")</code> or <code>showInPM("*")</code> which does NOT show the built-in properties.

Property Management Rules & Rules definition file

Built-in MySurvey 3D object types:

The full list of available values for the TYPE_MS property: "Annotation 3D", "Box", "Circle", "Cone", "Cylinder", "Dish", "Extruded polygon", "Mesh", "Photo", "Piping", "Point", "Point of view", "Polygon", "Polyline", "Pyramid", "Snapshot", "Sphere", "Rectangle", "Segment", "TextArea", and "Trihedron". This list is stored in an alias called TYPES_MS.

Example:

```
if prop("TYPE_MS") equals "Sphere" //If the provider's MySurvey 3D object type is a Sphere
    showInBubble("NAME") //Then show the item's name in the tooltip bubble
else
    showInPM("NAME") //Otherwise show the item's name in a PM field
end
```