URCap Working With Variables

Universal Robots A/S

Version 1.2.56

Abstract

As of URCap API version 1.2, child program nodes in a URCap program node contribution's sub tree can be configured. Part of this configuration can involve the use of variables. They can be used in built-in program nodes, such as the Assignment node, but also as part of an expression. Variables can also be used in the configuration of a URCap program node contribution.

Contents

1 Introduction

This document describes the purpose of variables in PolyScope and how to work with them through the URCap API. The use of variables will be slightly different than in the Java language itself, as there is a split between the identifier, i.e. the name of the variable, and its value. The identifier is used in the UI of PolyScope and script language, whereas the value only has meaning when the script code is executed.

2 Purpose of variables

The definition of a variable can roughly be stated as an identifier with an associated value. Its use in PolyScope can be many things depending on the context. The purpose of variables in the URCap API is mainly to be able to configure program nodes where a variable is a part of the configuration. This applies for built-in program nodes as well as URCap program nodes. Variables can also be used as part of an expression. Note that variables should not be used in an URCap installation node contribution, since it is a programming construct only used at runtime of a robot program.

Through the URCap API it is only possible to work with the variable as an identifier. Its value is not accessible through the URCap API as the value of a variable used in a program is determined on runtime (and thereby script code) only.

It is possible to give the variable an initial value through the URCap API, where the initial value is defined in the form of an expression that will be evaluated at runtime.

How to use a variable in a URCap program node naturally depends on the given context and what the URCap program node is trying to solve. However, a variable should be something the end user selects to work with through the UI and not as a temporary or internal construct in the script code.

3 Variable registration

PolyScope has a central service responsible for ensuring unique names for variables in use, as well as keeping track of the lifetime of the variables. This means that a URCap must register all variables it is using.

When configuring built-in nodes, a configuration using a variable is created and applied to the node. The variable is automatically registered when the configuration is applied to an existing node in the program tree or when a newly

created (and configured) node is inserted into the program tree.

For URCap nodes, the registration is done by putting the variable in the data model of the URCap node. This signals to PolyScope that this variable is in use by the URCap. If the URCap is no longer using the variable, the URCap must remove the variable from its data model. This happens by either calling the `remove()` method on the data model or by overwriting the value using the same key.

PolyScope will continuously keep track of all variables and who is using them. If a variable is no longer used by a built-in program node or stored in a URCap program node's data model, it will be removed from the system.

4 Creating a variable

A URCap can create a new variable by using the `VariableFactory` available through the `VariableModel` interface. Using the `createGlobalVariable(String)` method in the factory, it is possible to create a global variable accessible to the whole PolyScope program.

There is an method overload that also takes an expression as parameter, which defines the initial value of the variable. This can be useful for some applications, since it makes it possible to read from the newly created variable immediately without having to assign a value to it first.

The initial value is assigned before the actual program runs (in the preamble of the generated script code of the robot program). Note that this variable initialization will be accessible in the Init Variables program node (the very first node in the program tree), if this setting is enabled in Robot Program root node. Here the end user can edit or remove the expression. Variables should only be given an initial value, if it is strictly necessary that the variable is initialized before the actual robot program is executed.

In Listing 1 is an example of how to create a variable with an initial value and put it in the data model.

Listing 1: Creating a variable with an initial value

```
1 ExpressionBuilder expressionBuilder = api.
      getValueFactoryProvider().createExpressionBuilder();
2 Expression initialValue = expressionBuilder.append("0").build();
3 VariableFactory variableFactory = api.getVariableModel().
      getVariableFactory();
4 GlobalVariable variable = variableFactory.createGlobalVariable("
      myVar", initialValue);
5 model.set(SELECTED_VAR, variable);
```

4.1 Variable names

The name parameter of the create methods in the `VariableFactory` interface is no more than a suggested name, as the central service will provide a final name for the variable based on what names are in use at that time. The variable will not get its final name before being registered (i.e. put in the data model or used in a built-in program node). Remember that the variable is also subject for renaming by the end user. For this very reason, the name of a variable should not in any way be cached (e.g. in member variables) or used to refer to the variable.

If the suggested name is already used, an underscore and a running number will be appended to the name. Should this name also already be in use, the running number will be increased until the name is available. If the suggested name already ends with an underscore and a number, the number is simply turned into a running number if necessary. This means, that to create a series of variables, the suggested name should end with '_1'. This will create a contiguous series of named variables (unless one or more of the names are unavailable, in which case there will be a gap in the series where the name was already taken).

As previously described the newly created variable must be stored in the data model to make it available to other program nodes.

5 Resolving variables

When storing a variable in the data model, information about the type (e.g. global or installation) of the variable is also saved. When a program is loaded, any stored variables must be resolved before they can be used.

If the variable type is global and there are no name clashes with variables in the currently loaded installation, the variable is simply created and considered resolved. If there exists an installation variable with the same name, the end user is asked to resolve this by either renaming the global variable (automatically) or to switch the type to an installation variable. This means, that a stored variable can potentially change type after load.

Finally, if the variable type being loaded is an installation variable, the variable is resolved if there exists an installation variable in the currently loaded installation with the same name. If there is no such installation variable, the given URCap program node is undefined. See section 6.1 where this is explored in depth.

6 Using variables in a URCap program node

In the following sections a few common usages of variables in URCap program nodes will be described. Please also see the SDK samples *Cycle Counter* and *Idle Time* for further details of how to display variables, create new ones and use them in the script code generation.

6.1 Unresolved installation variables

If a URCap program node has an unresolved installation or feature variable (corresponding feature marked as variable) in its data model, the program node will by definition be undefined. This means the program node in PolyScope is colored yellow and the program cannot run. The end user must resolve the problem by either making the variable available again in the installation or change the variable selection in the URCap program node contribution.

The program node will become defined (also depending on its own internal state) if the variable becomes available again. This can happen either because the installation variable was created by the end user or a different installation containing that variable was loaded.

PolyScope will automatically ensure that a URCap program node using an unresolved variable becomes undefined (and defined should the variable become resolved again).

The `isResolvable()` method provided by the relevant variable URCap API interfaces can be used to determine, if a variable is resolved.

6.2 Rendering in UI elements

If the user interface of the URCap program node displays variables in a drop-down box UI component, they will be rendered (i.e. displayed) as in the built-in program nodes of PolyScope. This also means that the unresolved installation or feature variable mentioned in previous section will be rendered differently. The variable will be displayed in italics with a yellow border around it to indicate to the user that the variable is unresolved.

An unresolved variable is only rendered, if it is selected in drop-down box. Also note, that if unregistered variables are added to a drop-down box element, they will not be visible (nor selectable), since they do not represent a valid selection for the end user.

Listing 2 shows an example of how to add all program and installation variables sorted alphabetically to a drop-down box component.

Listing 2: Displaying program and installation variables in drop-down box element

```
1  ArrayList<Object> items = new ArrayList<Object>();
2  items.addAll(api.getVariableModel().get(new Filter<Variable>() {
3    @Override
4    public boolean accept(Variable element) {
5      return element.getType().equals(Variable.Type.GLOBAL) ||
           element.getType().equals(Variable.Type.VALUE_PERSISTED);
6    }
7  }));
8
9  Collections.sort(items, new Comparator<Object>() {
10   @Override
11   public int compare(Object o1, Object o2) {
12     if (o1.toString().toLowerCase().compareTo(o2.toString().
         toLowerCase()) == 0) {
13       //Sort lowercase/uppercase consistently
14       return o1.toString().compareTo(o2.toString());
15     } else {
16       return o1.toString().toLowerCase().compareTo(o2.toString()
           .toLowerCase());
17     }
18   }
19 });
20
21 //Insert at top after sorting
22 items.add(0, "Select␣variable");
23
24 comboVariables.setItems(items);
```

In the example, the drop-down box contains an initial '`Select variable`' item instructing the end user to select a variable. The code snippet is incomplete, since storing and retrieving the selection in the data model is missing. A method that executes when the select event is raised must be implemented in order to achieve this. When populating the drop-down box with variables (as shown in the example above) also set the selected item, if it is stored in the data model.

6.3 Variables in script code

When using a variable in the script code contribution, it is essential that the variable be retrieved from the data model and only used in methods with `Variable` typed parameters in the `ScriptWriter` interface. If it is necessary to use the variable in raw script code, the `getResolvedVariableName()` method in the `ScriptWriter` interface must be used to get a script safe/compatible resolved name.