URCap Screwdriver Driver

Universal Robots A/S

Version 1.13.0

Abstract

As of URCap API version 1.8, a driver framework is introduced for version 3.11.0/5.5.0 of PolyScope. This allows for adding functionality to PolyScope to support certain devices and having program nodes with UI as well as other contributions for free. This document will describe how to create a screwdriver driver with minimal Java code. Support for screwdriver drivers is introduced in Polyscope version 5.6.0 and is not available on CB3 robots.

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

Contents

1 Introduction **3**

2 General Information about URCap Development **3**

3 Building a Screwdriver Driver **3**
3.1 Mandatory Functionality . . . 4
3.2 Optional Functionality . . . 4
3.3 Capabilities (Optional) . . . 4
3.3.1 Program Selection . . . 5
3.3.2 Feed Screw . . . 5
3.3.3 Prepare To Start Screwdriver . . . 5
3.3.4 Operation Type . . . 6
3.4 Feedback Capabilities (Optional) . . . 6
3.4.1 Drive Screw OK . . . 6
3.4.2 Drive Screw Not OK . . . 6
3.4.3 Screwdriver Ready . . . 7
3.5 Custom User Configuration (Optional) . . . 7
3.5.1 Checkbox . . . 7
3.5.2 Combo box . . . 8
3.6 Other Configuration (Optional) . . . 9
3.6.1 TCP Contribution . . . 9
3.6.2 Tool I/O Interface . . . 10

4 Workflow **10**
4.1 PolyScope Integration . . . 10
4.1.1 Startup . . . 10
4.1.2 Creating an Installation . . . 11
4.1.3 Loading an Installation . . . 11
4.2 Script Code Generation . . . 11
4.2.1 Preamble . . . 11
4.2.2 Start Screwdriver . . . 11
4.2.3 Stop Screwdriver . . . 11
4.2.4 Running a Program with Screwdriving nodes . . . 12

5 Tips **12**

6 Advanced **13**
6.1 Backwards Compatibility . . . 13
6.2 Supporting Translations . . . 14

7 Samples **14**

Appendices **14**

A Activator **14**

B Simple Sample **14**

C Advanced Sample **16**

D Custom Sample **20**

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

1 Introduction

A driver in the context of PolyScope can be thought of as a way of having a specific device behaving in a uniform way when used in PolyScope.

In the case of a screwdriver driver, this means the driver will integrate into the builtin Screwdriving program node and installation node in PolyScope which will allow for programming and setting up the screwdriver, respectively. The screwdriver driver will be selectable in the Screwdriving installation node together with the default builtin user-defined screwdriver. When the end user selects the screwdriver driver, the screwdriver can be configured in the installation (if a custom installation UI is specified by the driver) and the screwdriver will also be the screwdriver used in the Screwdriving program node.

A screwdriver is defined as a device which can, as a minimum, start driving a screw and stop. How it performs these operations is defined using script code. A screwdriver can optionally register additional capabilities it might support. Doing this opens up the program node for further configuration by the user. This only applies for some of the capabilities, since some of them are not configurable by the user. The relevant user-defined parameters are accessible when generating script code for a given operation or capability.

In the following sections, starting from section 3. Building a Screwdriver Driver, a basic screwdriver driver will be constructed and then expanded to support various physical capabilities.

2 General Information about URCap Development

The URCap SDK includes the separate *URCap Tutorial Swing* document.

Here you can find general information about development of URCaps, such as how to the specify your URCap's compatibility with the different robot series (section 12.1. Robot Series Compatibility) which is mandatory, and how to deploy and install a URCap using Maven (section 6. Deployment with Maven).

The sections 7. Contribution of an Installation Node, 8. Contribution of a Program Node and 9. Contribution of a Daemon are not relevant, if your URCap is a "pure" screwdriver driver URCap that does not include any additional installation node contribution(s), program node contribution(s) or daemon contribution(s).

3 Building a Screwdriver Driver

PolyScope uses the OSGi framework to run. This means that to register a screwdriver driver, it is necessary to construct a bundle with an `Activator`. In the `Activator`, an implementation of the `ScrewdriverContribution` must be registered within the OSGi framework.

When PolyScope starts, it will pick up the screwdriver driver and construct an installation node. See appendix A. Activator for an example of this.

In the following subsections the implementation of the `ScrewdriverContribution` interface will be covered.

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

3.1 Mandatory Functionality

All methods in the `ScrewdriverContribution` interface are mandatory to implement, but the implementation for some of them can be left blank.

The `getTitle(Locale)` must return the title of the screwdriver. It is optional to use the `locale` parameter depending on whether translations are supported. The title is shown in the Screwdriving program node UI and the installation node UI.

The `generateStartScrewdriverScript(ScriptWriter, ScrewdriverParameters)` method must generate the necessary script code for starting the screwdriver using the `ScriptWriter` parameter. If any capabilities are registered, the configuration of these will be accessible through the `ScrewdriverParameters` parameter.

The `generateStopScrewdriverScript(ScriptWriter, ScrewdriverParameters)` method must generate the necessary script code for stopping the screwdriver using the the `ScriptWriter` parameter. If any capabilities are registered the configuration of these will be accessible through the parameter `ScrewdriverParameters`.

3.2 Optional Functionality

The `generatePreambleScript(ScriptWriter)` method can be used to generate any script code necessary for initializing the screwdriver. The script code will be added to the preamble section of the final script code generated for a robot program.

If no script code is necessary just leave this method blank.

The `configureContribution(ContributionConfiguration)` method can be used to configure the contribution itself using the `ContributionConfiguration` parameter. In the current version of the API only the logo for a screwdriver can be registered. This is shown on the Screwdriving program and installation node and must be a PNG or JPEG file.

The logo will automatically be scaled to fit in the mentioned places.

3.3 Capabilities (Optional)

A capability in the context of a screwdriver can be thought of as a physical property the screwdriver supports which in some cases can be configured by the end user.

The `configureScrewdriver(ScrewdriverConfiguration, ScrewdriverAPIProvider)` method can be used to register any capabilities the screwdriver may support using the `ScrewdriverConfiguration` interface. For access to PolyScope or robot domain specific information use the `ScrewdriverAPIProvider` parameter.

The majority of the capabilities are based on execution of script code. When these capabilities are registered, an implementation of the `ScriptCodeGenerator` interface must be supplied which will be responsible for generating the script code. PolyScope will ask the `ScriptCodeGenerator` implementation to generate the script code associated with a capability at appropriate times in the given context. Parameters relevant for the script code generation will be provided as input to the script code generation. The generated script code must return a boolean value indicating whether or not the execution of the capability was successful (i.e. a value of True or False).

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

If the screwdriver, for instance, supports automatic screwfeeding, the following code could be added:

```
1    screwdriverConfiguration.getScrewdriverCapabilities().
         registerFeedScrewCapability(createFeedScrewScriptCodeGenerator());
2
3    private ScriptCodeGenerator<FeedScrewParameters>
         createFeedScrewScriptCodeGenerator() {
4      return new ScriptCodeGenerator<FeedScrewParameters>() {
5        @Override
6        public void generateScript(ScriptWriter scriptWriter,
             FeedScrewParameters parameters) {
7          // Feed the screw
8          scriptWriter.appendLine("set_standard_digital_out(0, True)");
9          // Wait for the screw feeding to complete
10         scriptWriter.sleep(1.0);
11         // Check if the screw feeding succeeded
12         scriptWriter.appendLine("return get_standard_digital_in(1) == True");
13       }
14     };
15   }
```

The available screwdriver capabilities in the API are described in the following subsections.

3.3.1 Program Selection

This capability can be used if the screwdriver supports a selection of different programs, typically defined on an external control box for the screwdriver.

It is required that the generated script code must return a boolean value signifying whether or not the selection of the specified program was successful.

Registering this capability will display a combo box with the list of available screwdriver programs allowing the end user to select the program to use for the screwdriving operation. It will also enable the end user to add a Machine Error Handler to the Screwdriving program node, which provides the possibility of defining the processing steps to take, if the generated script code returns False.

An implementation of the `ScrewdriverProgramListProvider` interface must be supplied when registering the capability. When needed, PolyScope will request the `ScrewdriverProgramListProvider` implementation to provide the current list of screwdriver programs.

3.3.2 Feed Screw

This capability can be used if the screwdriver supports a screwfeeder.

It is required that the generated script code must return a boolean value signifying whether or not the screw feeding operation was successful.

Registering this capability will allow the end user to add a Machine Error Handler to the Screwdriving program node, which provides the possibility of defining the processing steps to take, if the generated script code returns False.

3.3.3 Prepare To Start Screwdriver

This capability can be used to perform actions for preparing the screwdriver for a screwdriving operation, i.e. before starting the screwdriver.

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

It is required that the generated script code must return a boolean value signifying whether or not the preparation of the screwdriver was successful.

Registering this capability will allow the end user to add a Machine Error Handler to the Screwdriving program node, which provides the possibility of defining the processing steps to take, if the generated script code returns False.

3.3.4 Operation Type

This capability specifies if the screwdriver supports selecting the direction to operate in.

Registering this capability will allow the end user to select the direction in the UI of the Screwdriving program node.

In the current implementation of the Screwdriving program node, this is only relevant when enabling the Follow the Screw option. If this capability is not registered, the direction is by default always tighten.

3.4 Feedback Capabilities (Optional)

A feedback capability in the context of a screwdriver can be thought of as a type of information, typically status, that the screwdriver is capable of informing PolyScope about.

As with non-feedback capabilities described in previous section, the feedback capabilities are based on execution of script code which is generated by an implementation of the interface `ScriptCodeGenerator`. The generated script code for a feedback capability must provide the information associated with the feedback capability through a boolean return value.

The available screwdriver feedback capabilities in the API are described in the following subsections.

3.4.1 Drive Screw OK

This capability can be used if the screwdriver can indicate when the ongoing screwdriving operation is done (ended in OK-state).

Registering this capability will allow the end user to add an OK success stop criteria to the Screwdriving node, which provides the possibility of defining the processing steps to take when the screwdriving operation was successful. This is done by selecting the OK success criteria in the Until node under the Screwdriving program node.

The generated script code must return a boolean value signifying whether or not the screwdriving operation was successful. Note that this return value should not be used to indicate that the operation failed, but simply that a success not was not (yet) detected. If the screwdriver can provide feedback on whether or not the the screwing operation failed, the Drive Screw Not OK feedback capability described in the following subsection must be used.

3.4.2 Drive Screw Not OK

This capability can be used if the screwdriver can indicate if something went wrong during an ongoing screwdriving operation (ended in Not OK-state).

Registering this capability will allow the end user to add a Not OK error stop criteria to the Screwdriving node, which provides the possibility of defining the processing steps to take if

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

the screwdriving operation fails. This is done by adding an Until node under the Screwdriving program node and selecting the Not OK error criteria in the Until node.

The generated script code must return a boolean value signifying whether or not the screwdriving operation failed, i.e. in the case of failure the return value must be True. Note that this return value should not be used to indicate that the operation succeeded, but simply that a failure not was not (yet) detected. If the screwdriver can provide feedback on whether or not the screwing operation was successful, the Drive Screw OK feedback capability described in the previous subsection must be used.

3.4.3 Screwdriver Ready

This capability can be used if the screwdriver can indicate when the screwdriver device is ready to operate (receive commands).

The generated script code must return a boolean value signifying whether or not the screwdriver is ready to operate.

Registering this capability will allow the end user to add a Machine Error Handler to the Screwdriving program node, which provides the possibility of defining the processing steps to take, if the generated script code returns False.

3.5 Custom User Configuration (Optional)

The `configureInstallation(CustomUserInputConfiguration, ...)` method can be used to register any custom user-defined input required for setting up the screwdriver. This could for instance be an IP-address or similar.

A number of widget types are supported, among other checkboxes, integer inputs and combo boxes.

All user inputs require an identifier (id) and a label. The id is the key used for storing the value in the underlying data model and the label is displayed next to the widget in the UI. It is important to never change the id of a user input, since this will break the persistence.

Non-user input widgets can also be added. These can be text labels (with an optional icon) and filler elements for controlling/grouping the layout of the UI.

The `UserInput` instance (or a sub type thereof) returned when a user input is registered should be stored locally in a class member field. The instance can be used when generating script code (either for the preamble section or a screwdriver operation) using the `getValue()` method which will return the value selected by the user.

It is possible to listen for changes to the user input in case this should trigger an action. This is done by calling the `setValueChangedListener(ValueChangedListener<T>)` method supplying a value change listener. The change listener will be called when the user changes the user input as well as when a new installation is loaded or created.

It is also possible to attach an error validator for some of the user inputs (generally the ones where the user enters the input) if need be.

3.5.1 Checkbox

To register a user input for a checkbox, use the following code:

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
BooleanUserInput checkbox = customConfiguration.registerBooleanInput("Id", "
    Label", false);
```

The third parameter (with the value of `false`) is a default value. This value is only used if the user has not yet changed the value of the user input. In case the user changed the value, the checkbox is initialized using the persisted value.

3.5.2 Combo box

Registering a combo box user input is a little different. There are two ways of registering a combo box input depending on whether the initial selection is valid or not (the invalid selection is a string value guiding the user to make a selection).

A combo box has an initial selection (either valid or invalid), a list of elements to show in the drop-down list, and finally an `ElementResolver` instance to correctly identify and display the elements in the UI. On top of this, it has the required id and label described previously.

The role of the element resolver is to help persisting the selection in the data model. Since the data model does not support persisting the entire selected object (which can be any type), the element resolver will return a `String` identifier (id) for the selected element. This id is persisted and when an installation is loaded, it is used to select the correct element.

Each element in the list must have a unique id, which must be constant over time and versions of the URCap, otherwise loading an installation may result in the combo box being unable to correctly initialize with the persisted selection. This could also happen if the contents of the list changed since the selection was made and persisted. In any case, the combo box's selection will be invalid, and a program using this screwdriver will be undefined and unable to run. The user must select a valid element in the combo box to fix the issue.

The implementation of the element resolver can optionally override the `getDisplayName(T)` method. This gives the screwdriver URCap the option to localize the UI or simply display a more meaningful name than the default implementation, which calls the `toString()` method on each element. The display name for the selected element is also stored in the data model, so it can be displayed in case the element is no longer in the list.

The code snippet in Listing 1 shows an example of the registration of a combo box with a custom element resolver.

Listing 1: Registering a combo box

```
1  SelectableUserInput<Mounting> comboBoxInput = customConfiguration.
       registerComboBoxInput("Id", "Mounting", "Select...", Arrays.asList(
       Mounting.values()), new ElementResolver<Mounting>() {
2    @Override
3    public String getId(Mounting element) {
4      return element.getId();
5    }
6
7    @Override
8    public String getDisplayName(Mounting element) {
9      return element.getDisplayName();
10   }
11 });
12
13 //-----------------
14
15
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
16  public enum Mounting {
17    TOOL_FLANGE("ToolFlange", "Normal"),
18    ANGLE45("Angle45", "45 degrees"),
19    ANGLE_NEG_45("Angle-45", "-45 degrees");
20
21    private String id;
22    private String name;
23
24    Mounting(String id, String name) {
25      this.id = id;
26      this.name = name;
27    }
28
29    public String getId() {
30      return id;
31    }
32
33    public String getDisplayName() {
34      return name;
35    }
36  }
```

The resulting combo box displayed in PolyScope can be seen in figure 1.

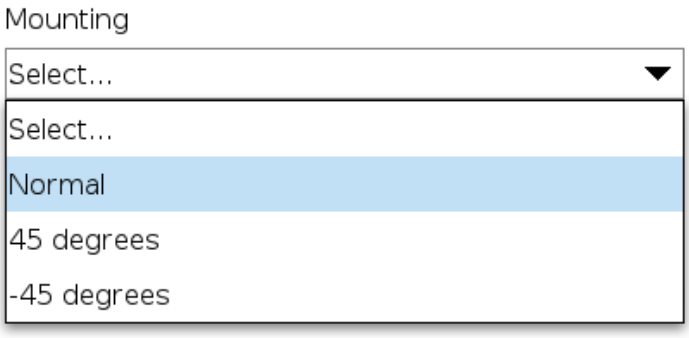

Figure 1: Combo box in PolyScope

3.6 Other Configuration (Optional)

This section describes configuration of the installation not directly related to the properties of the screwdriver itself. The configuration is performed in the `configureInstallation(...)` method using the different parameters as described in the following subsections.

3.6.1 TCP Contribution

It is possible to help the end user by preconfiguring the TCP of the screwdriver using the `TCPConfiguration` parameter of the `configureInstallation(...)` method. The TCP is added by calling the method `setTCP(String, Pose)` with a suggested name for the TCP and the pose for the offset of the TCP as input parameters. Use the `ScrewdriverAPIProvider` parameter to gain access to the pose factory capable of creating the needed pose for the TCP.

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

3.6.2 Tool I/O Interface

If the screwdriver is controlled and/or powered through the Tool I/O interface, a request to control the configuration of this interface must be made. This is done using the `SystemConfiguration` parameter.

Since some of the functionality is not available on CB3 robots, care must be taken to ensure the availability of the functionality before attempting to use it. To do this use the `CapabilityManager` interface to query the supported system capabilities (note that these capabilities are referring to the robot and are not capabilities of the screwdriver).

After ensuring the necessary system capabilities are available, a request to control the Tool I/O interface can be made. This happens through the `ControllableResourceModel` interface using the `requestControl(ToolIOInterfaceController)` method. The argument passed to the method will receive a callback if the end user assigns control to the screwdriver. When this happens the desired configuration of the Tool I/O interface can be applied.

Regardless of whether or not the user has granted the screwdriver the control of the Tool I/O interface, it is not permitted to control the settings directly via script code. The proper procedure is to have the user assign control, manipulate the settings in PolyScope in the callback described previously and let PolyScope generate the script code for applying the settings for the Tool I/O interface.

If read-only access to the Tool I/O interface is sufficient, simply use the `ResourceModel` interface provided by the aforementioned `SystemConfiguration` interface.

For more details on this, see the separate *Resource Control* document.

4 Workflow

Different parts of the implementation of the `ScrewdriverContribution` interface correspond to different phases in PolyScope. In the following subsections these parts and their phases will be described in detail.

4.1 PolyScope Integration

The phases PolyScope go through that will influence the screwdriver driver are:

- Startup
- Creation of an installation
- Loading an installation

4.1.1 Startup

When PolyScope starts, it will look for any registered contributions as previously described. This happens only on startup, so only the instance registered in the OSGi framework will be used. This one and only instance is used for the rest of the lifecycle of PolyScope. This means that no state other than the user input configuration should be maintained. The implementation should therefore rely on PolyScope keeping its state in the underlying data model.

During the startup phase, all methods in the `ScrewdriverContribution` interface will be called, except for the methods responsible for generating script code.

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

4.1.2 Creating an Installation

When creating an installation, the `configureScrewdriver(...)` and `configureInstallation(...)` methods are called again.

All user inputs will have the specified default values. Any value change listeners attached to the user inputs are not called (since technically there has been no change in value). This means that any state change that would happen when the default value is set, should also be the starting state of the screwdriver (e.g. if a checkbox determines whether a TCP should be offset with a camera ring, for instance, then the TCP pose offset should reflect the default value of the checkbox).

4.1.3 Loading an Installation

When loading an installation, the same methods are called as when creating a new installation. However, the default value is not being used, if the user has previously modified the value. In this case, the user input will have the persisted value restored and a call to the attached value change listener will be made, so any dependent state can reflect the value (e.g. same scenario as when creating an installation, only here the call will be made and the TCP pose offset can be adjusted according to the actual value of the checkbox).

4.2 Script Code Generation

The main part of a screwdriver driver is its script code generation. This determines how the screwdriver is initialized and operated. The script code generation consists of three separate parts:

- Preamble
- Start Screwdriver operation
- Stop Screwdriver operation

4.2.1 Preamble

The implementation of the `generatePreambleScript(ScriptWriter)` method should use the `ScriptWriter` parameter to initialize the screwdriver in such a way that each consecutive operation of the screwdriver (e.g. start screwdriver and stop screwdriver) can be executed without further initialization between each operation. The implementation of this method can reference any custom user input registered, if necessary, but there are no capability parameters as these are tied to the screwdriver operation or capability execution.

4.2.2 Start Screwdriver

The `generateStartScrewdriverScript(ScriptWriter, ScrewdriverParameters)` method is called when the script code for a screwing operation should be generated. The implementation of this method can also reference any registered custom user input if necessary. It will also have access to the parameter for any relevant capability registered. It is not legal to retrieve a value for a capability not registered. Doing so will throw an exception.

4.2.3 Stop Screwdriver

The `generateStopScrewdriverScript(ScriptWriter, ScrewdriverParameters)` method is called when the script code for a stop screwdriver operation should be generated. The implementation of this method can also reference any custom user input registered if necessary. It will also have access to the parameter for any relevant capability registered. It is not legal to retrieve a value for a

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

capability not registered. Doing so will throw an exception. It should be noted that you should make sure to stop the screwdriver when pausing/stopping a program, since this is not done automatically by Polyscope.

4.2.4 Running a Program with Screwdriving nodes

When running a program with Screwdriving program nodes, the following order of execution applies for the script code generated by the screwdriver driver (for each program node):

1. Screwdriver Ready (if capability is supported) (see 3.4.3. Screwdriver Ready)
2. Program Selection (if capability is supported) (see 3.3.1. Program Selection)
3. Feed Screw (if capability is supported) (see 3.3.2. Feed Screw)
4. Prepare To Drive Screw (if capability is supported) (see 3.3.3. Prepare To Start Screwdriver)
5. Start Screwdriver
6. Threaded polling of (if selected as success/failure option):
   (a) Drive Screw OK (if capability is supported) (see 3.4.1. Drive Screw OK)
   (b) Drive Screw Not OK (if capability is supported) (see 3.4.2. Drive Screw Not OK)
7. When the operation completed by any condition:
   (a) Stop Screwdriver

5 Tips

In general, all method implementations should execute fast to provide a smooth PolyScope user experience. This is especially true for custom error validators attached to user inputs, since these get called each time the input changes (e.g. when the user enters something, the error validator is called for each keystroke).

Take an IP address input as an example. The user input itself validates by default that what the user inputs is in fact a valid IP address, but it could seem natural to ensure that the entered IP address is the address used by the device by pinging a server. Since this is time consuming, it might delay the user feedback in PolyScope. Consider solving it by adding a status label informing the user of whether the server on the entered IP address is responding. This way, a change listener could run a separate thread that checks whether the server is accessible on the IP address, and once this check is over update the status label.

The program list provider for the Program Selection capability (see section 3.3.1. Program Selection) must also transfer the current list of screwdriver program fast when PolyScope requests it. This is important, since it ensures a good user experience when the UI can be updated with the available programs to select from as quickly as possible.

Script code should also run fast. This means that the script code for the screwdriver should be the bare minimum needed. This will speed up both the generation itself, but also the compilation and execution performed by the controller.

The script code generated for Drive Screw OK (see section 3.4.1. Drive Screw OK) and Drive Screw Not OK feedback (see section 3.4.2. Drive Screw Not OK) capabilities will be called frequently while the Screwdriving node is executing, so it is absolutely vital that this script code performs well.

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

6 Advanced

In this section, a few advanced topics will be covered.

6.1 Backwards Compatibility

If the screwdriver driver evolves over time and any required custom user input changes, backwards compatibility must be considered. As a running example a checkbox determining whether or not a camera ring is mounted will be used.

In the first version of the screwdriver driver this is, as mentioned, simply a checkbox. In the next version of the screwdriver driver the manufacturer now supports a force/torque ring and the camera ring. The checkbox has to be deprecated, since the natural custom user input would now be a combo box. However, it is possible to help the user by preselecting the proper replacement in the combo box. The way to handle this is shown in the code snippet in listing 2.

If the checkbox was checked, the proper replacement is the 'Camera Ring' element in the combo box. On the other hand, if the checkbox was unchecked, the proper replacement is the 'No offset' element in the combo box. This is shown in the helper method `convertValue(Boolean)`.

When deprecating a user input, it will no longer be shown in the UI and its persisted value will be removed once the user saves the installation (but the value replacing it will now be stored). Note that the id of the user input being deprecated must not be reused for the new user input.

Listing 2: Deprecating a user input

```
1  ...
2  @Override
3  public void configureInstallation(CustomUserInputConfiguration
       customConfiguration, SystemConfiguration systemConfiguration,
       TCPConfiguration tcpConfiguration, ScrewdriverAPIProvider apiProvider) {
4    BooleanUserInput camera = customConfiguration.registerBooleanInput("Camera",
         "Camera␣ring␣mounted", false);
5    Boolean cameraValue = customConfiguration.deprecateUserInput(camera);
6
7    customConfiguration.registerPreselectedComboBoxInput("TCPOffset", "Select␣
       additional␣mounting", convertValue(cameraValue), Arrays.asList(TCPOffset
       .values()), new ElementResolver<TCPOffset>() {
8      @Override
9      public String getId(TCPOffset element) {
10       return element.toString();
11     }
12   });
13 }
14
15 private TCPOffset convertValue(Boolean cameraValue) {
16   return cameraValue ? TCPOffset.CAMERA_RING : TCPOffset.NO_OFFSET;
17 }
18 ...
19
20 //--------------------
21
22 public enum TCPOffset {
23   NO_OFFSET,
24   CAMERA_RING,
25   FT_SENSOR
26 }
27 ...
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

6.2 Supporting Translations

Supporting translations in a screwdriver driver only applies to custom user inputs and the title of the screwdriver. The rest is handled by PolyScope.

For the title, the `Locale` is provided as parameter to the `getTitle(Locale)` method in the interface `ScrewdriverContribution`.

For the custom user inputs, the `Locale` can be found in the `SystemSettings` interface which is accessed through the `SystemAPI` interface provided by the `ScrewdriverAPIProvider` interface. The only thing that should be translated are labels and the return value of the `getDisplayName(T)` method of an `ElementResolver` implementation (if using combo boxes). Never translate the id of a user input nor change them from version to version of the screwdriver driver as this will break the persistence of the settings for the screwdriver driver.

7 Samples

The appendices show listings with the code of an activator, a simple screwdriver, an advanced screwdriver and a screwdriver with a custom UI in the Screwdriving installation node for setting up the screwdriver. For further details of the included screwdriver driver samples, please see the 10.2. Driver Samples section in the separate *URCap Tutorial Swing* document.

Appendices

A Activator

Listing 3: Activator class registering a simple screwdriver driver

```
1  package com.ur.urcap.examples.driver.screwdriver.simplescrewdriver;
2
3  import com.ur.urcap.api.contribution.driver.screwdriver.
       ScrewdriverContribution;
4  import org.osgi.framework.BundleActivator;
5  import org.osgi.framework.BundleContext;
6
7  public class Activator implements BundleActivator {
8
9    @Override
10   public void start(final BundleContext context) {
11     context.registerService(ScrewdriverContribution.class, new
           SimpleScrewdriver(), null);
12   }
13
14   @Override
15   public void stop(BundleContext context) {
16
17   }
18 }
```

B Simple Sample

Listing 4: SimpleScrewdriver class containing main functionality of the simple screwdriver driver

```
1  package com.ur.urcap.examples.driver.screwdriver.simplescrewdriver;
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
2
3  import com.ur.urcap.api.contribution.driver.general.tcp.TCPConfiguration;
4  import com.ur.urcap.api.contribution.driver.general.userinput.
       CustomUserInputConfiguration;
5  import com.ur.urcap.api.contribution.driver.screwdriver.
       ContributionConfiguration;
6  import com.ur.urcap.api.contribution.driver.screwdriver.ScrewdriverAPIProvider
       ;
7  import com.ur.urcap.api.contribution.driver.screwdriver.
       ScrewdriverConfiguration;
8  import com.ur.urcap.api.contribution.driver.screwdriver.
       ScrewdriverContribution;
9  import com.ur.urcap.api.contribution.driver.screwdriver.ScrewdriverParameters;
10 import com.ur.urcap.api.contribution.driver.screwdriver.SystemConfiguration;
11 import com.ur.urcap.api.domain.script.ScriptWriter;
12
13 import javax.swing.ImageIcon;
14 import java.util.Locale;
15
16
17 public class SimpleScrewdriver implements ScrewdriverContribution {
18
19   @Override
20   public String getTitle(Locale locale) {
21     return "Simple␣Screwdriver";
22   }
23
24   @Override
25   public void configureContribution(ContributionConfiguration configuration) {
26     configuration.setLogo(new ImageIcon(getClass().getResource("/logo/logo.png
           ")));
27   }
28
29   @Override
30   public void configureScrewdriver(ScrewdriverConfiguration
         screwdriverConfiguration, ScrewdriverAPIProvider screwdriverAPIProvider)
          {
31     // Intentionally left empty
32   }
33
34   @Override
35   public void configureInstallation(CustomUserInputConfiguration
         configurationUIBuilder,
36                     SystemConfiguration systemConfiguration,
37                     TCPConfiguration tcpConfiguration,
38                     ScrewdriverAPIProvider apiProvider) {
39     // Intentionally left empty
40   }
41
42   @Override
43   public void generatePreambleScript(ScriptWriter scriptWriter) {
44     // Intentionally left empty
45   }
46
47   @Override
48   public void generateStartScrewdriverScript(ScriptWriter scriptWriter,
         ScrewdriverParameters parameters) {
49     scriptWriter.appendLine("set_standard_digital_out(1,␣True)");
50   }
51
52   @Override
53   public void generateStopScrewdriverScript(ScriptWriter scriptWriter,
         ScrewdriverParameters parameters) {
54     scriptWriter.appendLine("set_standard_digital_out(2,␣False)");
55   }
56 }
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

C Advanced Sample

Listing 5: AdvancedScrewdriver class containing main functionality of the advanced screwdriver driver

```
1  package com.ur.urcap.examples.driver.screwdriver.advancedscrewdriver;
2
3  import com.ur.urcap.api.contribution.driver.general.script.ScriptCodeGenerator
       ;
4  import com.ur.urcap.api.contribution.driver.general.tcp.TCPConfiguration;
5  import com.ur.urcap.api.contribution.driver.general.userinput.
       CustomUserInputConfiguration;
6  import com.ur.urcap.api.contribution.driver.screwdriver.
       ContributionConfiguration;
7  import com.ur.urcap.api.contribution.driver.screwdriver.ScrewdriverAPIProvider
       ;
8  import com.ur.urcap.api.contribution.driver.screwdriver.
       ScrewdriverConfiguration;
9  import com.ur.urcap.api.contribution.driver.screwdriver.
       ScrewdriverContribution;
10 import com.ur.urcap.api.contribution.driver.screwdriver.ScrewdriverParameters;
11 import com.ur.urcap.api.contribution.driver.screwdriver.SystemConfiguration;
12 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       DriveScrewNotOKParameters;
13 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       DriveScrewOKParameters;
14 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       FeedScrewParameters;
15 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       PrepareToStartScrewdriverParameters;
16 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       ProgramSelectionParameters;
17 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       ScrewdriverCapabilities;
18 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       ScrewdriverFeedbackCapabilities;
19 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       ScrewdriverReadyParameters;
20 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       screwdriverprogram.ScrewdriverProgram;
21 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       screwdriverprogram.ScrewdriverProgramList;
22 import com.ur.urcap.api.contribution.driver.screwdriver.capability.
       screwdriverprogram.ScrewdriverProgramListProvider;
23 import com.ur.urcap.api.domain.script.ScriptWriter;
24
25 import javax.swing.ImageIcon;
26 import java.util.Arrays;
27 import java.util.List;
28 import java.util.Locale;
29
30
31 public class AdvancedScrewdriver implements ScrewdriverContribution {
32
33   private static final int SCREWDRIVER_READY_STATUS_DIGITAL_INPUT = 0;
34   private static final int FEED_SCREW_STATUS_DIGITAL_INPUT = 1;
35   private static final int DRIVE_SCREW_OK_DIGITAL_INPUT = 2;
36   private static final int DRIVE_SCREW_NOK_DIGITAL_INPUT = 3;
37
38   private static final int PROGRAM_SELECTION_DIGITAL_OUTPUT = 0;
39   private static final int REQUEST_FEED_SCREW_DIGITAL_OUTPUT = 1;
40   private static final int DRIVE_SCREW_DIGITAL_OUTPUT = 2;
41
42   // Represents a program for the screwdriver
43   private enum Program {
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
44      LOW("Program_1_id", "Program␣1"),
45      HIGH("Program_2_id", "Program␣2");
46
47      private final String id;
48      private final String displayName;
49
50      Program(String id, String displayName) {
51        this.id = id;
52        this.displayName = displayName;
53      }
54
55      String getId() {
56        return id;
57      }
58
59      String getDisplayName() {
60        return displayName;
61      }
62    }
63
64
65    @Override
66    public String getTitle(Locale locale) {
67      return "Advanced␣Screwdriver";
68    }
69
70    @Override
71    public void configureContribution(ContributionConfiguration configuration) {
72      configuration.setLogo(new ImageIcon(getClass().getResource("/logo/logo.png
          ")));
73    }
74
75    @Override
76    public void configureScrewdriver(ScrewdriverConfiguration
          screwdriverConfiguration, ScrewdriverAPIProvider screwdriverAPIProvider)
           {
77      ScrewdriverCapabilities capabilities = screwdriverConfiguration.
          getScrewdriverCapabilities();
78      capabilities.registerProgramSelectionCapability(
          createScrewdriverProgramListProvider(),
79                              createProgramSelectionScriptCodeGenerator());
80      capabilities.registerFeedScrewCapability(
          createFeedScrewScriptCodeGenerator());
81      capabilities.registerPrepareToStartScrewdriverCapability(
          createPrepareToStartScrewdriverScriptCodeGenerator());
82      capabilities.registerOperationTypeCapability();
83
84      ScrewdriverFeedbackCapabilities feedbackCapabilities =
          screwdriverConfiguration.getScrewdriverFeedbackCapabilities();
85      feedbackCapabilities.registerScrewdriverReadyCapability(
          createScrewdriverReadyScriptCodeGenerator());
86      feedbackCapabilities.registerDriveScrewOKCapability(
          createDriveScrewOKScriptCodeGenerator());
87      feedbackCapabilities.registerDriveScrewNotOKCapability(
          createDriveScrewNotOKScriptCodeGenerator());
88    }
89
90    private ScrewdriverProgramListProvider createScrewdriverProgramListProvider
          () {
91      return new ScrewdriverProgramListProvider() {
92        @Override
93        public void populateList(ScrewdriverProgramList screwdriverProgramList)
              {
94          screwdriverProgramList.addAll(retrieveScrewdriverProgramList());
95        }
96      };
97    }
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
98
99    // This method "simulates" retrieving the current list of screwdriver
          program (typically from the control box for
100   // the screwdriver).
101   private List<ScrewdriverProgram> retrieveScrewdriverProgramList() {
102     List<ScrewdriverProgram> availableScrewdriverPrograms = Arrays.asList(
          createScrewdriverProgram(Program.LOW),
103                                         createScrewdriverProgram(Program.HIGH)
                                                ;
104     return availableScrewdriverPrograms;
105   }
106
107   private ScrewdriverProgram createScrewdriverProgram(final Program program) {
108     return new ScrewdriverProgram() {
109       @Override
110       public String getId() {
111         return program.getId();
112       }
113
114       @Override
115       public String getDisplayName() {
116         return program.getDisplayName();
117       }
118     };
119   }
120
121   private ScriptCodeGenerator<ProgramSelectionParameters>
          createProgramSelectionScriptCodeGenerator() {
122     return new ScriptCodeGenerator<ProgramSelectionParameters>() {
123       @Override
124       public void generateScript(ScriptWriter scriptWriter,
            ProgramSelectionParameters parameters) {
125         ScrewdriverProgram screwdriverProgram = parameters.
              getScrewdriverProgram();
126         boolean programSelectionOutputValue = screwdriverProgram.getId().
              equals(Program.HIGH.getId());
127
128         scriptWriter.appendLine(generateSetterScriptForDigitalOutput(
              PROGRAM_SELECTION_DIGITAL_OUTPUT, programSelectionOutputValue));
129         // Note: In a real application, this line should return a boolean
              indicating whether or not selecting
130         // the program succeeded.
131         scriptWriter.appendLine("return␣True");
132       }
133     };
134   }
135
136   // We will request feed screw, wait 0.5 second and then return a status
137   private ScriptCodeGenerator<FeedScrewParameters>
          createFeedScrewScriptCodeGenerator() {
138     return new ScriptCodeGenerator<FeedScrewParameters>() {
139       @Override
140       public void generateScript(ScriptWriter scriptWriter,
            FeedScrewParameters parameters) {
141         scriptWriter.appendLine(generateSetterScriptForDigitalOutput(
              REQUEST_FEED_SCREW_DIGITAL_OUTPUT, true));
142         // Simulate feeding a screw to the screwdriver
143         scriptWriter.sleep(0.5);
144         // Check if the feeding the succeeded
145         scriptWriter.appendLine("return␣" +
              generateGetterScriptForDigitalInput(
              FEED_SCREW_STATUS_DIGITAL_INPUT) + "␣==␣True");
146       }
147     };
148   }
149
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
150     private String generateSetterScriptForDigitalOutput(int digitalOutputId,
            boolean value) {
151       String valueString = value ? "True" : "False";
152       return "set_standard_digital_out(" + digitalOutputId + "," + valueString +
              ")";
153     }
154
155     private String generateGetterScriptForDigitalInput(int digitalInputId) {
156       return "get_standard_digital_in(" + digitalInputId + ")";
157     }
158
159     private ScriptCodeGenerator<PrepareToStartScrewdriverParameters>
            createPrepareToStartScrewdriverScriptCodeGenerator() {
160       return new ScriptCodeGenerator<PrepareToStartScrewdriverParameters>() {
161         @Override
162         public void generateScript(ScriptWriter scriptWriter,
              PrepareToStartScrewdriverParameters
              prepareToStartScrewdriverParameters) {
163           // Simulate preparing the screwdriver
164           scriptWriter.sleep(0.5);
165           // Note: In a real application, this line should return a boolean
                indicating whether or not preparing
166           // the screwdriver succeeded.
167           scriptWriter.appendLine("return␣True");
168         }
169       };
170     }
171
172     private ScriptCodeGenerator<ScrewdriverReadyParameters>
            createScrewdriverReadyScriptCodeGenerator() {
173       return new ScriptCodeGenerator<ScrewdriverReadyParameters>() {
174         @Override
175         public void generateScript(ScriptWriter scriptWriter,
              ScrewdriverReadyParameters screwdriverReadyParameters) {
176           scriptWriter.appendLine("return␣" +
                generateGetterScriptForDigitalInput(
                SCREWDRIVER_READY_STATUS_DIGITAL_INPUT) + "␣==␣True");
177         }
178       };
179     }
180
181     private ScriptCodeGenerator<DriveScrewOKParameters>
            createDriveScrewOKScriptCodeGenerator() {
182       return new ScriptCodeGenerator<DriveScrewOKParameters>() {
183         @Override
184         public void generateScript(ScriptWriter scriptWriter,
              DriveScrewOKParameters driveScrewOKParameters) {
185           scriptWriter.appendLine("return␣" +
                generateGetterScriptForDigitalInput(DRIVE_SCREW_OK_DIGITAL_INPUT)
                + "␣==␣True");
186         }
187       };
188     }
189
190     private ScriptCodeGenerator<DriveScrewNotOKParameters>
            createDriveScrewNotOKScriptCodeGenerator() {
191       return new ScriptCodeGenerator<DriveScrewNotOKParameters>() {
192         @Override
193         public void generateScript(ScriptWriter scriptWriter,
              DriveScrewNotOKParameters driveScrewNotOKParameters) {
194           scriptWriter.appendLine("return␣" +
                generateGetterScriptForDigitalInput(DRIVE_SCREW_NOK_DIGITAL_INPUT)
                 + "␣==␣True");
195         }
196       };
197     }
198
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
199     @Override
200     public void configureInstallation(CustomUserInputConfiguration
            configurationUIBuilder,
201                                       SystemConfiguration systemConfiguration,
202                                       TCPConfiguration tcpConfiguration,
203                                       ScrewdriverAPIProvider apiProvider) {
204       configurationUIBuilder.setDescriptionText("This␣is␣an␣example␣of␣a␣
              Screwdriver␣Driver␣supporting␣all␣capabilities");
205     }
206
207     @Override
208     public void generatePreambleScript(ScriptWriter scriptWriter) {
209       // Intentionally left empty
210     }
211
212     @Override
213     public void generateStartScrewdriverScript(ScriptWriter scriptWriter,
            ScrewdriverParameters parameters) {
214       scriptWriter.appendLine(generateSetterScriptForDigitalOutput(
              DRIVE_SCREW_DIGITAL_OUTPUT, true));
215     }
216
217     @Override
218     public void generateStopScrewdriverScript(ScriptWriter scriptWriter,
            ScrewdriverParameters parameters) {
219       scriptWriter.appendLine(generateSetterScriptForDigitalOutput(
              DRIVE_SCREW_DIGITAL_OUTPUT, false));
220     }
221   }
```

D Custom Sample

Listing 6: CustomScrewdriver class containing main functionality of the custom screwdriver driver

```
1  package com.ur.urcap.examples.driver.screwdriver.customscrewdriver;
2
3  import com.ur.urcap.api.contribution.driver.general.tcp.TCPConfiguration;
4  import com.ur.urcap.api.contribution.driver.general.userinput.
       CustomUserInputConfiguration;
5  import com.ur.urcap.api.contribution.driver.general.userinput.enterableinput.
       StringUserInput;
6  import com.ur.urcap.api.contribution.driver.general.userinput.selectableinput.
       BooleanUserInput;
7  import com.ur.urcap.api.contribution.driver.screwdriver.
       ContributionConfiguration;
8  import com.ur.urcap.api.contribution.driver.screwdriver.ScrewdriverAPIProvider
       ;
9  import com.ur.urcap.api.contribution.driver.screwdriver.
       ScrewdriverConfiguration;
10 import com.ur.urcap.api.contribution.driver.screwdriver.
       ScrewdriverContribution;
11 import com.ur.urcap.api.contribution.driver.screwdriver.ScrewdriverParameters;
12 import com.ur.urcap.api.contribution.driver.screwdriver.SystemConfiguration;
13 import com.ur.urcap.api.domain.script.ScriptWriter;
14 import com.ur.urcap.api.domain.value.Pose;
15 import com.ur.urcap.api.domain.value.PoseFactory;
16 import com.ur.urcap.api.domain.value.simple.Angle;
17 import com.ur.urcap.api.domain.value.simple.Length;
18
19 import javax.swing.ImageIcon;
20 import java.util.Locale;
21
22
23 public class CustomScrewdriver implements ScrewdriverContribution {
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
24
25    private static final String SHOW_POPUP_INPUT_ID = "showPopupInputID";
26    private static final String IP_ADDRESS_INPUT_ID = "IPAddressInputID";
27
28    private BooleanUserInput showPopup;
29    private StringUserInput ipAddress;
30
31    @Override
32    public String getTitle(Locale locale) {
33      return "Custom␣Screwdriver";
34    }
35
36    @Override
37    public void configureContribution(ContributionConfiguration configuration) {
38      configuration.setLogo(new ImageIcon(getClass().getResource("/logo/logo.png
            ")));
39    }
40
41    @Override
42    public void configureScrewdriver(ScrewdriverConfiguration
          screwdriverConfiguration, ScrewdriverAPIProvider screwdriverAPIProvider)
           {
43      // Intentionally left empty
44    }
45
46    @Override
47    public void configureInstallation(CustomUserInputConfiguration
          configurationUIBuilder,
48                      SystemConfiguration systemConfiguration,
49                      TCPConfiguration tcpConfiguration,
50                      ScrewdriverAPIProvider apiProvider) {
51      addScrewdriverTCP(tcpConfiguration, apiProvider.getPoseFactory());
52      customizeInstallationScreen(configurationUIBuilder);
53    }
54
55    private void addScrewdriverTCP(TCPConfiguration tcpConfiguration,
          PoseFactory poseFactory) {
56      Pose tcpOffset = poseFactory.createPose(0, 80, 120, 0, 0, 0, Length.Unit.
            MM, Angle.Unit.RAD);
57      tcpConfiguration.setTCP("My_Screwdriver", tcpOffset);
58    }
59
60    private void customizeInstallationScreen(CustomUserInputConfiguration
          configurationUIBuilder) {
61      this.showPopup = configurationUIBuilder.registerBooleanInput(
            SHOW_POPUP_INPUT_ID, "Show␣popup", true);
62      this.ipAddress = configurationUIBuilder.registerIPAddressInput(
            IP_ADDRESS_INPUT_ID, "IP␣Address", "127.0.0.1");
63    }
64
65    @Override
66    public void generatePreambleScript(ScriptWriter scriptWriter) {
67      // Intentionally left empty
68    }
69
70    @Override
71    public void generateStartScrewdriverScript(ScriptWriter scriptWriter,
          ScrewdriverParameters parameters) {
72      if (showPopup.getValue()) {
73        scriptWriter.appendLine("popup(\"IP␣Address:␣" + ipAddress.getValue() +
              "\",blocking=True)");
74      }
75      scriptWriter.appendLine("#start␣screwdriver␣at␣IP␣Address:␣" + ipAddress.
            getValue());
76    }
77
78    @Override
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved

```
79    public void generateStopScrewdriverScript(ScriptWriter scriptWriter,
          ScrewdriverParameters parameters) {
80      scriptWriter.appendLine("#stop screwdriver at IP Address: " + ipAddress.
            getValue());
81    }
82  }
```

Copyright © 2009-2021 by Universal Robots A/S, all rights reserved