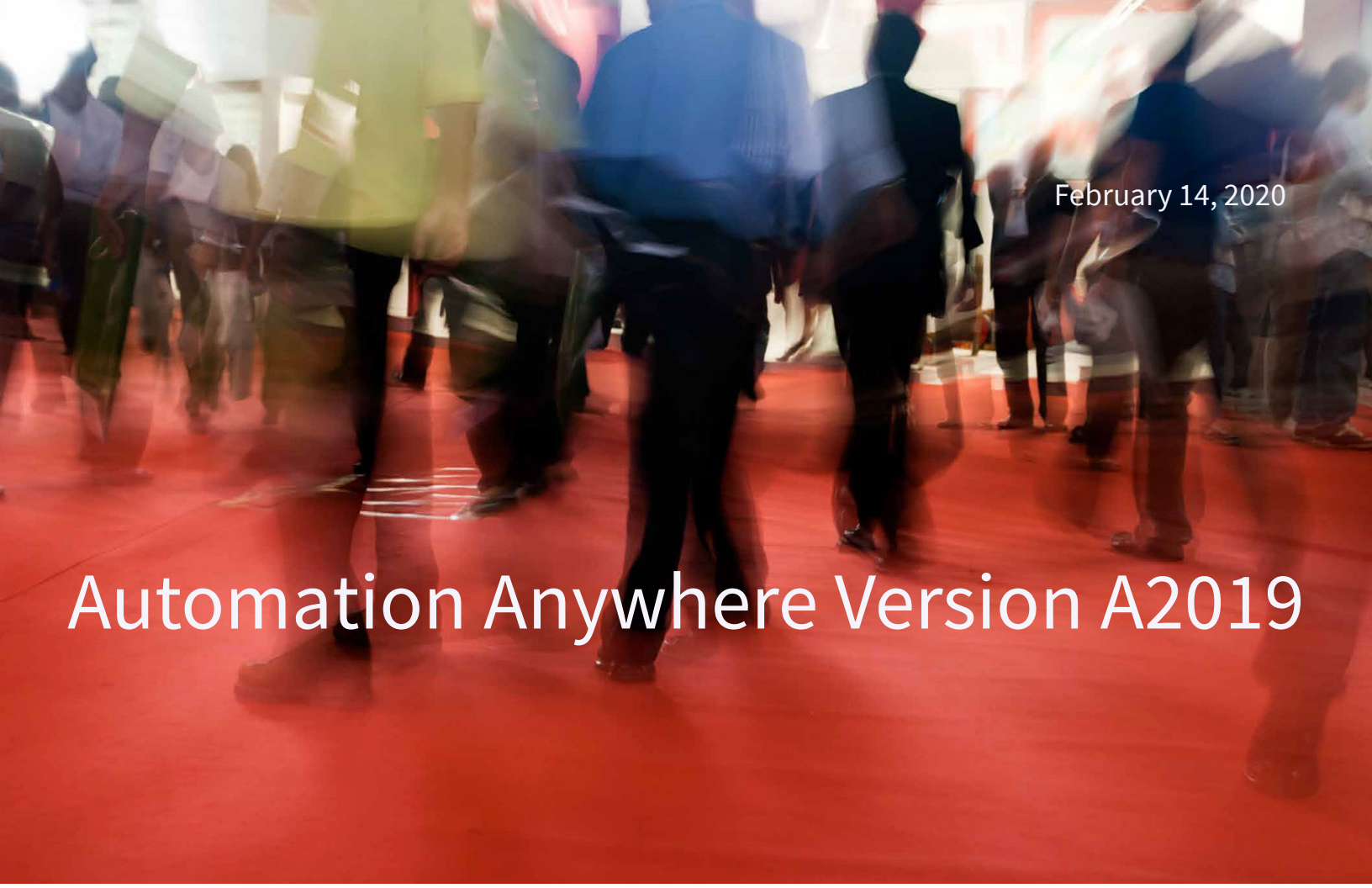

February 14, 2020

Automation Anywhere Version A2019

Legal Notices

Content

Automation Anywhere Enterprise Package Development Kit

This document explains the requirements and process for creating and uploading an action package to your Enterprise Control Room.

Packages are Java Archive (JAR) files containing the executable third-party applications used to create bots. Actions are available in the Enterprise Control Room under the Action Panel.

The Automation Anywhere Package Development Kit provides detailed instructions for users to independently develop custom actions and upload and manage packages in their Enterprise Control Room.

The SDK includes sample code and supporting files for Java developers to create and validate custom actions.

January 2020, Release (A2019.09)

- SDK Demo Package: A2019.09-packageSDK-1.0.0.zip
- Documentation: A2019.09-package-annotations-javadoc.zip

November 2019 Release (A2019.08):

- SDK Demo Package: A2019.08-packageSDK-1.0.0.zip
- Documentation: A2019.08-package-annotqations-javadoc.zip

October 2019 Release (A2019.07):

- SDK Demo Package: A2019.07-Package-Sdk-1.0.0.zip
- Documentation: A2019.07-package-annotations-javadoc.zip

August 2019 Release (A2019.06):

- SDK Demo Package: A2019DemoPackage.zip
- Documentation: A2019-package-annotations-javadoc.zip

For detailed release notes for the SDK Packages, see Enterprise A2019 Package Development Kit Release Notes.

Click a title to read details about each task in the process.

Setting up the Java project
: Set up an Integrated Develop Environment (IDE) for Java, including Automation Anywhere custom annotations to enable the development of action packages that can be uploaded to your Enterprise Control Room.

Standard coding practices and guidelines for developing packages
: This topic covers standard coding practices and guidelines that help to ensure the development of high quality packages.

Develop a sample package
: Develop your own package and upload it to an Enterprise Control Room to provide custom actions for bots.

How to examples
: This section contains code examples and explanations about how to code some basic bot capabilities.

Annotations
: This section provides reference information about the annotations used to create Automation Anywhere packages.

Build and test a demo package and bot
: This practical how to section demonstrates that creating, changing, and managing packages allow you to customize actions and efficiently manage packages for all Enterprise Control Room users.

Related reference
Enterprise A2019 Package Development Kit Release Notes

Setting up the Java project

Set up an Integrated Develop Environment (IDE) for Java, including Automation Anywhere custom annotations to enable the development of action packages that can be uploaded to your Enterprise Control Room.

Prerequisites

A working knowledge of Java and Gradle is needed in order to successfully build an action package. You need the following software and file:

- Java Developer Kit (JDK) 11
- Java IDE
  - Eclipse
  - Community edition of IntelliJ
- Gradle plug-in v.5.*.* in the IDE.
- Automation Anywhere Bot Agent installed and connected to valid Enterprise Control Room account
- Download and extract the following zip archive files:
  January 2020, Release (A2019.09)
  - SDK Demo Package: A2019.09-packageSDK-1.0.0.zip
  - Documentation: A2019.09-package-annotations-javadoc.zip

  November 2019 Release (A2019.08):
  - SDK Demo Package: A2019.08-packageSDK-1.0.0.zip
  - Documentation: A2019.08-package-annotqations-javadoc.zip

  October 2019 Release (A2019.07):
  - SDK Demo Package: A2019.07-Package-Sdk-1.0.0.zip
  - Documentation: A2019.07-package-annotations-javadoc.zip

  August 2019 Release (A2019.06):
  - SDK Demo Package: A2019DemoPackage.zip
  - Documentation: A2019-package-annotations-javadoc.zip

Initial set up of your IDE is important so that you have the correct environment for creating Automation Anywhere action packages.
Important: The listed prerequisites are recommendations for use with all the sample code and instructions included in this package development kit.

Procedure

1. Extract the content of the associated sample zip files to a folder you can access from your IDE.
2. Import the content in any java IDE of your choice as a Gradle project.
3. Edit the settings.gradle and change the root project name to something meaningful to you.

Tip: The settings.gradle file is included in the A2019DemoPackage.zip file.

4. Go to src > main > resources > package.template and change the package name and related information to something meaningful.

Original package.template	Updated package.template

<pre>{
  "name": "A2019DemoPackage",
  "label": "A2019DemoPackage",
  "description": "Provides actio
ns for
  A2019DemoPacka
ge operations.",
  "group": "",
  "artifactName": "",
  "packageVersion": "",
  "codeVersion": "",
  "commands": []
}</pre>
	<pre>{
	  "name": "YourPackageName",
	  "label": "Appropriate label",
	  "description": "Meaningful des
	cription for the actions
	  contained in t
	he package.",
	  "group": "",
	  "artifactName": "",
	  "packageVersion": "",
	  "codeVersion": "",
	  "commands": []
	}</pre>

Tip: The package template file controls the following names and labels of your package.

- "name" is the JAR file name
  Package file naming conventions:
  - No spaces
  - No special characters
- "label" is what appears in the Action panel of your Enterprise Control Room

Related concepts
Standard coding practices and guidelines for developing packages
How to examples
Annotations
Build and test a demo package and bot
Related tasks
Develop a sample package

Standard coding practices and guidelines for developing packages

This topic covers standard coding practices and guidelines that help to ensure the development of high quality packages.

Testing
Ensure high quality code. Write sufficient unit tests and integration tests for your package.

Icons
: Set proper icon for your package.

Dependencies
: Embed all the dependencies in your package JAR. Load the dependencies at run time by extracting them to a temporary location. Be sure to clean the temporary location after the dependencies are loaded.

Dependent JAR files
: Add dependent JAR files under dependencies in in the build.gradle file as implementation so that the the dependant JAR files are packaged.

```
. . .
dependencies {
                compileOnly name: 'command-annotations'
                compileOnly name: 'bot-runtime'
                compileOnly name: 'bot-api'
                implementation name: 'i18n-api'
              implementation name: 'mydependentjavafile.jar'
                apt name: 'command-processor'
                compileOnly group: 'org.apache.logging.log4j', name: 'log4
j-core', version: "$loggerVersion"
                testImplementation "org.testng:testng:$testNgVersion"
                testImplementation name: 'bot-runtime'
                testImplementation name: 'bot-api'
        }
. . .
```

Add new actions to exiting package
: When adding new actions to existing package, make sure to do clean before packaging. Its always a good practice to do clean build - gradlew.bat clean build shadowJar

Error messages
: Provide meaningful error messages.

- Do throw meaningful error messages. For example, in local language using i18n APIs with `BotCommandException`, throw a new exception `BotCommandException(MESSAGES.getString("Run.Exception.InvalidWorkingDirPath"))`.
- Do not throw generic error messages, such as `ex.message`.

Basic validation
: Use the validation annotation rules such as `@NotEmpty` included with this development kit. Do not add basic validations for your code. Read Validation annotations for more details.

Loops
: Avoid long running loops in your code. Long running loops can cause high CPU usage, leading to errors such as, "Bot is unresponsive."

Add logging
: Use the default log4J logger provided in the bot run time framework. Do not add your own logger. See the sample code for more details.

Related concepts
How to examples
Annotations
Related tasks
Setting up the Java project
Develop a sample package

Develop a sample package

Develop your own package and upload it to an Enterprise Control Room to provide custom actions for bots.

Prerequisites

Download and extract the contents from A2019DemoPackage.zip. This package contains the necessary source code for the sample package.

You need to have a project created in a Java IDE. For details about setting up a project, read Setting up the Java project.

The following high-level tasks provide the basic workflow for creating a package.

Procedure

1. Create a java class.
   This class is the action you plan to publish in your package.
   Important: It is required that the class support the default constructor.
2. Add required business logic to the class.
   The following are the supported return types:
   - Void: Use this return type is your action does not return any value.
   - Value: Use this return type if your action returns any type of value.
3. Annotate the class with BotCommand and CommandPkg annotations to make the class eligible to be converted to an action.
4. Annotate the variable that accept values with Idx and Pkg.
5. Annotate the entry method with the Execute annotation.
6. From the action prompt, run `gradlew.bat clean build shadowJar`.
   The JAR file created from the build is located in build/lib.
7. From the Enterprise Control Room on the Bots > Package, click the Add package icon to upload JAR file.

Next steps

From the Enterprise Control Room on the Bots > Package page, click the Add package icon to upload the JAR file.

Tip: To upload a package to a Enterprise Control Room, you need Upload package permission. Read details about how to add a package to an Enterprise Control Room: Add packages to the Enterprise Control Room.
Related concepts
Standard coding practices and guidelines for developing packages

How to examples
Annotations
Related tasks
Setting up the Java project

How to examples

This section contains code examples and explanations about how to code some basic bot capabilities.

Return a value from an action
: Set the following properties on CommandPkg to store the action's output in a variable.

Expose an action as a property
: An action can be exposed as property if it does not accept any parameter. This can be done by setting the following properties on CommandPkg.

Add a condition in a custom package for If condition
: Add conditions in a custom package.

Add an iterator in a custom package for Loop action
: Add an iterator in your package for Loop action.

Add debug logs of custom packages to bot_launcher.log file
: Logging can be added using log4j. The dependency is already added in the sample build.gradle.

Handle session in custom package
: Actions need to extract the required session from the SessionsMap by the session name.

There are three types of actions:

- Command\Action (default choice)
- Iterator
- Condition

Any action class supports only one method as an entry point. Annotate all parameters of the entry point method with Idx.
CAUTION: If you do not provide a public setter to member variables with Inject, compilation errors occur.
Related concepts
Standard coding practices and guidelines for developing packages
Annotations
Related tasks
Setting up the Java project
Develop a sample package

Return a value from an action

Set the following properties on CommandPkg to store the action's output in a variable.

Action return values

return_type
: Defines the return type of the action. It usually matches the entry method's return type.

return_required
: Makes the assignment operation compulsory when true.

return_label
: The UI label when asking for the variable to store the value in.

```
@BotCommand
@CommandPkg(label = "Uppercase", name = "uppercase", description="Converts the
source string to upper case.",
icon = "uppercase.svg", node_label="Convert {{sourceString}} to upper case| an
d assign the result to {{returnTo}}|",
return_type=DataType.STRING, return_required = true, return_label="Assign the o
utput to variable",
property_name="uppercase", property_description="Converts the string to upper c
ase", property_type=DataType.STRING,
property_return_type=DataType.STRING) public class UpperCase {

    @Execute
    public Value<String> convert(
            @Idx(index = "1", type=TEXT)
            @Pkg(label="Source string")
            @NotEmpty
            String sourceString){
        return new StringValue(sourceString.toUpperCase());
    }
}
}
```

Related concepts
How to examples

Expose an action as a property

An action can be exposed as property if it does not accept any parameter. This can be done by setting the following properties on CommandPkg.

Action property values

property_name
: The name of the property, unique at action level, in auto-complete box this name would appear.

property_description
: A description of the property.

property_type
: The data type on which property operates, only if the type matches, the property will be appear in the auto-complete box.

property_return_type
: The data type for what property returns. If this type does not match with the field type where it is used, there will be validation error.

```
@BotCommand
@CommandPkg(label = "Uppercase", name = "uppercase", description="Converts the
source string to upper case.",
icon = "uppercase.svg", node_label="Convert {{sourceString}} to upper case| an
d assign the result to {{returnTo}}|",
return_type=DataType.STRING, return_required = true, return_label="Assign the o
utput to variable",
property_name="uppercase", property_description="Converts the string to upper c
ase", property_type=DataType.STRING,
property_return_type=DataType.STRING) public class UpperCase {

    @Execute
    public Value<String> convert(
            @Idx(index = "1", type=TEXT)
            @Pkg(label="Source string")
            @NotEmpty
            String sourceString){
        return new StringValue(sourceString.toUpperCase());
    }
}
```

Related concepts
How to examples

Add a condition in a custom package for If condition

Add conditions in a custom package.

Create condition values in an Action

- To create a condition set commandType property of BotCommand annotation with value as Condition.
- To define the entry method of the condition use the annotation ConditionTest.


```
@BotCommand(commandType = Condition)
@CommandPkg(label = "File exists", name = "fileExists",
        description = "Checks the file exists condition.",
        node_label = "file exists at {{sourceFilePath}}", icon = "")
public class Exist extends AbstractCondition {
    @ConditionTest
    public boolean test(@Idx(index = "1", type = FILE) @LocalFile @Pkg(label =
"File path") @NotEmpty String sourceFilePath,
                        @Idx(index = "2", type = NUMBER) @Pkg(label = "How lon
g you would like to wait for this condition
                        to be true?(Seconds)",
                                default_value = "0", default_value_type = DataT
ype.NUMBER)
                        @GreaterThanEqualTo("0") @LessThanEqualTo("99999") @Not
Empty @NumberInteger Double waitTimeout) {

        // Logic to check for the condition goes here

    }
}
```

Related concepts
How to examples

Add an iterator in a custom package for Loop action

Add an iterator in your package for Loop action.

Add an iterator to a Loop action

- To create an iterator, set commandType property of BotCommand annotation with value as Iterator.
- There are two methods required by iterator, and they are defined by HasNext, and Next annotations.

```
@BotCommand(commandType = BotCommand.CommandType.Iterator)
@CommandPkg(name = "loop.iterators.files",
        label = "For each file in folder",
        node_label = "for each file and assign file name and extension to {{ret
urnTo}}",
```



```
        description = "Iterator for each file in folder.",
        return_type = DataType.DICTIONARY,
        return_sub_type = DataType.STRING,
        return_required = true,
        return_description = "Note: Access the 'name' key to access file name a
nd 'extension'
        key to access the file extension.",
        return_label = "Assign file name and extension to this variable")
public class FileLoop extends AbstractCommandFileIterator {

    @Idx(index = "1", type = AttributeType.TEXT)
    @Pkg(label = "Folder path")
    @Inject
    @NotEmpty
    private String folderPath;

    @HasNext
    public boolean hasNext() {
        return getFileIterator(folderPath).hasNext();
    }

    @Next
    public Value<?> next() {
        Map<String, Value> returnValueMap = new HashMap<>();

        FileIterator fileIterator = getFileIterator(folderPath);
        String fileName = fileIterator.getNext();

        returnValueMap.put(FILE_NAME, new StringValue(fileIterator.getFileName(
fileName)));
        returnValueMap.put(EXTENSION, new StringValue(fileIterator.getExtensio
n(fileName)));

        return new DictionaryValue(returnValueMap);

    }
```



```
    public void setFolderPath(String folderPath) {
        this.folderPath = folderPath;
    }
}
```

Related concepts
How to examples

Add debug logs of custom packages to bot_launcher.log file

Logging can be added using log4j. The dependency is already added in the sample build.gradle.

```
@BotCommand
@CommandPkg(label = "Copy to",icon="assigntoclipboard.svg" ,name = "assignToCli
pboard", description
 = "Accepts user input or a variable and assigns it to Clipboard", node_label="
{{value}}")
public class AssignToClipboard {

    private static Logger logger = LogManager.getLogger(AssignToClipboard.class
);

    @Execute
    public static void assign(@Idx(index = "1", type = TEXT) @Pkg(label = "Valu
e") @NotEmpty String
    value) {

        logger.trace("Assigning '{}' value to clipboard.", value);
    }
}
```

Related concepts
How to examples

Handle session in custom package

Actions need to extract the required session from the SessionsMap by the session name.

SessionsMap instance can be received using the Sessions attribute. The annotation can only be applied to class field and a corresponding public setter is expected. The variable must be of type Map<String,Object>.

```
@BotCommand
@CommandPkg(label = "Start session", name = "startSession", description = "Star
t new session",
icon = "pkg.svg", node_label = "start session {{sessionName}}|") public class S
tart {

    @Sessions
    private Map<String, Object> sessions;

    @Execute
    public void start(@Idx(index = "1", type = TEXT) @Pkg(label = "Session name
",
    default_value_type = STRING, default_value = "Default") @NotEmpty String se
ssionName) {

        // Check for existing session
        if (sessions.containsKey(sessionName))
            throw new BotCommandException(MESSAGES.getString("xml.SessionNameIn
Use", sessionName));

        // Do some operation

        // Create new session
        sessions.put(sessionName, new Session(operation));

    }

    public void setSessions(Map<String, Object> sessions) {
        this.sessions = sessions;
    }
}
```

```
@BotCommand
@CommandPkg(label = "End session", name = "endSession", description = "End sess
```



```
ion", icon =
"pkg.svg", node_label = "End session {{sessionName}}|")
public class EndSession {



    @Sessions
    private Map<String, Object> sessions;

    @Execute
    public void end(
            @Idx(index = "1", type = TEXT) @Pkg(label = "Session name", default
_value_type = STRING,
            default_value = "Default") @NotEmpty String sessionName) {

        sessions.remove(sessionName);

    }

    public void setSessions(Map<String, Object> sessions) {
        this.sessions = sessions;
    }
}
```

Related concepts
How to examples

Annotations

This section provides reference information about the annotations used to create Automation Anywhere packages.

Creation and function annotations
: List of the available creation and function annotations.

Validation annotations
: Validates annotated strings and values used in your Java code.

Related concepts
Standard coding practices and guidelines for developing packages
How to examples
Related tasks
Setting up the Java project
Develop a sample package

Creation and function annotations

List of the available creation and function annotations.

Annotation	Description
BotCommand	Makes the type eligible to be treated as an `action`. You can define 3 types of actions commandType property. • Command\Action • Condition • Iterator Examples: • `@BotCommand(commandType = BotCommand.CommandType.Iterator)` • `@BotCommand(commandType = Condition)`
CommandPkg	Makes the type eligible for creation of action package.json. This annotation must be used with BotCommandc to take effect. Pkg would participate in the activity only when this annotation is present. Example: `@CommandPkg(label = "Create", name = "createFile", description = "Creates a file", node_label = "{{filePath}}", icon = "file.svg")`
ConditionTest	Method annotated with this annotation will participate in the execution of Condition. This annotation can only be used when the BotCommand has commandType set as Condition. Exactly one method needs to be annotated when BotCommand annotation is present on the type. Failure to do so will result in compilation error.
Execute	Method annotated with this annotation will participate in the execution of BotCommand. Exactly one method needs to be annotated when BotCommand annotation is present on the type. Failure to do so will result in compilation error.

Annotation	Description
	Example: `@Execute public void create(@Idx(index = "1", type = FILE) @LocalFile @Pkg(label = "File", description = "e.g. C:\\MyDoc\\MyFile.doc") @NotEmpty String filePath, @Idx(index = "2", type = CHECKBOX) @Pkg(label = "Over write an existing file") @NotEmpty Boolean isOverwrite) { createFile(filePath , isOverwrite); }`

GlobalSessionContext	Can only be applied to member variables and fetches the GlobalSessionContext through a setter.
	Example:
	`@com.automationanywhere.commandsdk.annotations.Global SessionContext`
	`private GlobalSessionContext globalSessionContext ;`
	
	`public void setSessionMap(Map < String, Object > sessionMap) {`
	`this.sessionMap = sessionMap;`
	`}`
	
	`public void setGlobalSessionContext(com.automatio nanywhere.bot.service.GlobalSessionContext globalSess ionContext) {`
	`this.globalSessionContext = globalSessionContext ;`
	`}`

HasNext	Method annotated with this annotation will participate in the execution of Iterator.

<table>
<tr><th>Annotation</th><th>Description</th></tr>
<tr><td></td><td>This annotation can only be used when the BotCommand has commandType set as Iterator.

Requires annotation Next to be present.

Exactly one method needs to be annotated when BotCommand annotation is present on the type. Failure to do so will result in compilation error.</td></tr>
<tr><td>Idx</td><td>Makes the annotated element part of hierarchy utilized for code and resource generation. In other words without this annotation no BotCommand related element annotations is processed.</td></tr>
<tr><td>Idx.Option</td><td>An option represents the elements that would play in the hierarchy, but lend the values to the parents.

Examples:

• RADIO

<pre>@Idx(index = "1", type = RADIO, options = {
    @Idx.Option(index = "1.1", pkg = @Pkg(node_label = "[[Delay.delayType.1.1.node_label]]", label = "[[Delay.delayType.1.1.label]]", value = REGULAR)),
    @Idx.Option(index = "1.2", pkg = @Pkg(node_label = "[[Delay.delayType.1.2.node_label]]", label = "[[Delay.delayType.1.2.label]]", value = RANDOM))
})
@Pkg(label = "[[Delay.delayType.label]]", default_value = "REGULAR", default_value_type = DataType.STRING)
@Inject
private String delayType;</pre>
• SELECT

<pre>@Idx(index = "2", type = SELECT, options = {
@Idx.Option(index = "2.1", pkg = @Pkg(label = "[[LaunchWebsite.browser.2.1.label]]", value = "DEFAULT")),</pre></td></tr>
</table>

Annotation	Description

	`@Idx.Option(index = "2.2", pkg = @Pkg(label = "[[LaunchWebsite.browser.2.2.label]]", value = "INTERNET_EXPLORER")),`
	`@Idx.Option(index = "2.3", pkg = @Pkg(label = "[[LaunchWebsite.browser.2.3.label]]", value = "FIREFOX")),`
	`@Idx.Option(index = "2.4", pkg = @Pkg(label = "[[LaunchWebsite.browser.2.4.label]]", value = "CHROME"))`
	`}) @Pkg(label = "[[LaunchWebsite.browser.label]]", default_value = "DEFAULT", default_value_type = DataType.STRING) @NotEmptyStringbrowser)`

Inject	Makes an element eligible for injection into the annotated type's object. The injection is setter based so a corresponding setter in the type is mandatory.Injected values would from BotCommand parameter map using the name provided in Idx.
Next	Method annotated with this annotation will participate in the execution of Iterator. This annotation can only be used when the BotCommand has commandType set as Iterator. Requires annotation HasNext to be present. Exactly one method needs to be annotated when BotCommand annotation is present on the type. Failure to do so will result in compilation error.
Pkg	Makes an element participate in creation of package.json. This annotation is ignored when Idx is not present.

Related concepts
Annotations

Validation annotations

Validates annotated strings and values used in your Java code.

Annotation	Description
CodeType	The MIME-type of the code to format.
CredentialOnly	Can only accept a credential value, no plain string allowed.
Equals	Validates the given String is equal to annotated String variable.
FileExtension	Validates the annotated String value ends with the supported extension type.
GreaterThan	Validates that the annotated number variable value is always greater than given numeric value.
GreaterThanEqualTo	Validates that the annotated number variable value is always greater than or equal to given numeric value.
LessThan	Validates that the annotated number variable value is always less than given numeric value.
LessThanEqualTo	Validates that the annotated number variable value is always less than or equal to given numeric value.
LocalFile	Can only accept local paths and no file expression.
MatchesRegex	Validates the annotated String value matches the for given regular expression.

NotEmpty	Validates and throws exception when the annotated variable value is null.
	`@Execute`
	`public Value<Double>length(@Idx(index="1",`
	`type=TEXT)@Pkg(label="Source string")@NotEmpty Str`
	`ing`
	`sourceString){}`

NotEquals	Validates that the given String is not be equal to annotated String variable.
NotMatchesRegex	Validates the annotated String value does not match the for given regular expression.
NumberInteger	Ensures the UI accepts only integers not double for the annotated variable value.
RepositoryFile	Can only accept repository paths and no file expression.
VariableNotPackage	Cannot choose a variable from this package.
VariablePackage	Can only choose a variable from this package.
VariableSubType	The variable subtype that must match
VariableType	The variable type that must match
VariableUserDefined	Can only choose user-defined a variable.

Related concepts
Annotations

Build and test a demo package and bot

This practical how to section demonstrates that creating, changing, and managing packages allow you to customize actions and efficiently manage packages for all Enterprise Control Room users.

Here is a list of all the necessary tasks to create a package, add the package to your Enterprise Control Room, and verify your work in a bot. Complete the listed tasks in order.
Tip: Click the title of each step to go the detailed task.

Step 1 Update related workflow and build files
Follow the detailed steps for updating workflow and build files for this project using your integrated development environment (IDE).
Step 2 Choose your favorite IDE
You can compile a package from the IDE of your choice. Here are two possible ways you can compile a package:

Compile a demo JAR file from the Eclipse UI
Use Eclipse to compile a demo JAR file that you can add as a package to your Enterprise Control Room.
Compile a demo JAR file from the command line
Compile the demo Java code provided with this software development kit.

Step 3 Add your demo package to an Enterprise Control Room
Users with Upload package permission can add packages to the Enterprise Control Room for use by all Bot Creators.
Step 4 Create a demo bot with the demo package
Create a bot using the demo package to verify the actions that were created.
Step 5 Change the Java file used to create the package JAR file
Modify and compile the Java code used to create a package to fix issues and create new functionality.
Step 6 Upload new demo package
Package management allows you to upload package updates. The new package has the same name, but a different version number.
Step 7 Update the demo bot with the updated package
Update bots to use specific package versions.

Update related workflow and build files

Follow the detailed steps for updating workflow and build files for this project using your integrated development environment (IDE).

Prerequisites

Complete all the steps for project setup detailed in Setting up the Java project.

This task shows you how to update the appropriate build and workflow files.

Procedure

1. Open the project "A2019DemoPakcage" that you created in Setting up the Java project.
   You can find the project file in the directory where you extract the zip files to, for example `c:\A2019DemoPackage`.

2. From inside the project, Open the settings.gradle file.
3. Replace the project name with `A2019DemoPackageFirstnameLastname`.
   If your name is John Developer it would look like this, `A2019DemoPackageJohnDeveloper`.
4. Open src > main > resources > package.template.
5. Update the name, label, and description values.

Original package.template	Updated package.template

<pre>{
  "name": "A2019DemoPackage",
  "label": "A2019DemoPackage",
  "description": "Provides actions for A2019DemoPackage operations.",
  "group": "",
  "artifactName": "",
  "packageVersion": "",
  "codeVersion": "",
  "commands": []
}</pre>
	<pre>{
	  "name": "A2019DemoPackageFirstnameLastname",
	  "label": "A2019DemoPackageFirstnameLastname",
	  "description": "A2019DemoPackageFirstnameLastname",
	  "group": "",
	  "artifactName": "",
	  "packageVersion": "",
	  "codeVersion": "",
	  "commands": []
	}</pre>

Tip: The package template file controls the following names and labels of your package.
- "name" is the JAR file name
  Package file naming conventions:
  - No spaces
  - No special characters
- "label" is what appears in the Action panel of your Enterprise Control Room

6. Save the changes.

Next steps

After you have setup the build files, you need to compile the demo Java code, Compile a demo JAR file from the command line.

Compile a demo JAR file from the Eclipse UI

Use Eclipse to compile a demo JAR file that you can add as a package to your Enterprise Control Room.

Prerequisites

Before starting this task complete the steps in Update related workflow and build files.

Build a package file using a Gradle project in the Eclipse IDE.

Procedure

1. Import the A2019DemoPackage as a Cradle project, File > Import > Cradle > Existing Cradle Project and click Finish.
2. From the Cradle Tasks tab, go to <your project> > build and run the following tasks in order.
   a) <your project> > build > clean
   a) <your project> > build > build
3. From the Cradle Tasks tab, go to <your project. > shadow and run the shadowJar task.

   Your compiled package file is located in file:\ \A2019DemoPackage\build\libs\. The package file has named after your project name (<your project>-1.0.0.jar).

Next steps

To add your custom package to your Enterprise Control Room follow the instructions in Add packages to the Enterprise Control Room.
Related concepts
Build and test a demo package and bot

Compile a demo JAR file from the command line

Compile the demo Java code provided with this software development kit.

Prerequisites

Before starting this task complete the steps in Update related workflow and build files.

Procedure

1. Open a terminal window and go to where the gradlew.bat file is located.
   ...\A2019DemoPackage > gradlew.bat
2. In the terminal window, type `gradlew.bat clean build shadowJar`, and press Enter. Here is an example of what you see:

```
> . . .\A2019DemoPackage>gradlew.bat clean build shadowJar


> Task :compileJava
Note: Starting hierarchy discovery for 'com.automationanywhere.botcommand.
demo.Concatenate'
Note: Starting non-hierarchical element discovery for 'com.automationanywh
ere.botcommand.demo.Concatenate'
Note: Starting hierarchy discovery for 'com.automationanywhere.botcommand.
demo.Uppercase'
```



```
Note: Starting non-hierarchical element discovery for 'com.automationanywh
ere.botcommand.demo.Uppercase'
Note: Starting Command Java generator...
Note: Starting Json generator...
Note: Generating command json for Concatenate
Note: Generating command json for Uppercase

> Task :commandCodeGen
mergeJsonFiles: updatePackage: group com.automationanywhere , artifactNam
e A2019DemoPackageFirstnameLastname ,
packageVersion 1.0.0-20190816-101906
```

The compiled file is located in file:\ \A2019DemoPackage\build\libs\.

Next steps

To add your custom package to your Enterprise Control Room follow the instructions in Add packages to the Enterprise Control Room.
Related concepts
Build and test a demo package and bot

Add your demo package to an Enterprise Control Room

Users with Upload package permission can add packages to the Enterprise Control Room for use by all Bot Creators.

Prerequisites

Before you can upload a package, you need valid user login credentials with Upload package permission for the Enterprise Control Room you are adding the package to.

Procedure

1. From the Bots > Packages page, click the Add package icon.
2. Browse to the location of the package to add.
   Packages are Java Archive (JAR) files that contain actions used to create bots.
3. Select the package to add, and click Upload package.
4. On the Bots > Packages > Confirm package page, there are 3 options:

   Reject
   : Stops the upload process.

   Accept, enable and set as default
   : Uploads and enables the selected package, setting to the default package for the Enterprise Control Room.

Accept and enable
: Uploads and enables the package, but the package is not set as the default package. Bot Creators need to specifically select non-default packages to use them for creating bots.

Next steps

After successfully uploading your demo package, create a bot to test the actions you just created. For detailed step about how to create a bot, read Create a demo bot with the demo package

Create a demo bot with the demo package

Create a bot using the demo package to verify the actions that were created.

Prerequisites

Here are the minimum prerequisites for building this demo bot:

- Access to a Control Room
- User credentials with AAE_Basic permission
- Your local host (workstation) is a registered device in the Control Room
- Ensure that the demo package A2019DemoPackageFirstnameLastname is available in the Enterprise Control Room

This task uses the following actions and components:

- Uppercase (demo package)
- Message box
- Variables and credentials

Procedure

1. Go to Bots > My bots and click the My Task Bot icon.
2. Type
   `MyDemoBot1`
   in the Name field.
3. Click Create & Edit.
4. Expand A2019DemoPackageFirstnameLastname and double Uppercase.
5. Type
   `hello world, go be great!`
   in all lower case letters.
6. Create the variable
   `vMyDemoVar1`
   .
7. Click Apply.
8. Add a Message box and insert the variable vMyDemoVar1 in the Enter the message to display field.
9. Click Apply and Save.
10. Click the Run icon.
    A message box with "GO BE GREATE!" in all upper case letters is displayed. The custom action Uppercase converted all the letters from lower case letters to upper case letters.

Next steps

The task, Change the Java file used to create the package JAR file, gives instruction on how to modify the Uppercase action to convert all upper case letters to lower case letters.

Change the Java file used to create the package JAR file

Modify and compile the Java code used to create a package to fix issues and create new functionality.

Prerequisites

Procedure

1. Open the project "A2019DemoPakcage."
2. From inside the project, open src/main/java > Uppercase.java > Uppercase.
3. Change the function from upper case to lower case.

Original function	Updated function

`. . .`
 	
`String result = sourceString.toUpperCase();`
 	
`return new StringValue(result);`
 	
`}`
 	
`}`
	`. . .`
	
	`String result = sourceString.toLowerCase();`
	
	`return new StringValue(result);`
	
	`}`
	
	`}`

4. Save the changes and re-compile the package.

Next steps

You can now upload the changed package to the Enterprise Control Room.

Upload new demo package

Package management allows you to upload package updates. The new package has the same name, but a different version number.

Prerequisites

You need AAE_Basic permissions to create and edit bots.

Procedure

1. From the Bots > Packages page, click the Add package icon.
2. Browse to the location of the package to add.
   Packages are Java Archive (JAR) files that contain actions used to create bots.
3. Select the package to add, and click Upload package.
4. On the Bots > Packages > Confirm package page, click Accept, enable and set as default.

Next steps

You can select specific packages to be used from within a bot. Read detailed steps about managing packages for specific bots in Update the demo bot with the updated package task.

Update the demo bot with the updated package

Update bots to use specific package versions.

Prerequisites

- Access to the bot created in the task Create a demo bot with the demo package.
- AAE_Basic permission.

Procedure

1. Go to Bots > My bots and double click MyDemoBot1, the demo bot you created in an earlier task.
2. Click the vertical eclipses in the upper right corner and click Packages.
3. Expand the row for the package A2019DemoPackageFirstnameLastnam.
4. From the dropdown list of package versions, select the Defaultversion.
   Because you added the updated package as the default version, you are selecting the new version of the package you created.
5. Click Change Version and Save.
6. Go to Bots > My bots and double click MyDemoBot1.
7. Click A2019DemoPackageFirstnameLastnam and type
   ```
   HELLO WORLD, GO BE GREAT!
   ```
   in the Source string field.
8. Click Apply and Save.
9. Click the Run icon.
   The message box displayed by the bot displays "hello world, go be great!" This verifies that the action from the updated package is being used.

Enterprise A2019 Package Development Kit Release Notes

These release notes describe new features, changed features, fixed features, security fixes, deprecated features, and known limitations in the Enterprise A2019 Package Development Kit.

Enhancements A2019.09

Feature	Description
Bot run-time libraries	Bundled latest bot run-time libraries for A2019.09 Package SDK.

January 2020, Release (A2019.09)

- SDK Demo Package: A2019.09-packageSDK-1.0.0.zip
- Documentation: A2019.09-package-annotations-javadoc.zip

Enhancements A2019.08

Feature	Description
Bot run-time libraries	Bundled latest bot run-time libraries for A2019.08 Package SDK.
Properties support	Extended support for properties.
Comments expanded and improved	Added more comments to sample commands to help use SDK.
Comment field formatting	Added text color and background color to comment fields.

November 2019 Release (A2019.08):

- SDK Demo Package: A2019.08-packageSDK-1.0.0.zip
- Documentation: A2019.08-package-annotqations-javadoc.zip

Enhancements A2019.07

Feature	Description
Bot run-time libraries	The A2019.07 bot run-time libraries are bundled in the SDK package.
CREDENTIAL attribute	We provide support for credential attributes that require input from action screens.
Localized error messages and actions UI text	Enables developing packages with i18n error messages and localized (l10n) actions UI with localized text.

Feature	Description
Java Development Kit 11	The A2019.07 Package Development Kit supports JDK 11.

October 2019 Release (A2019.07):

- SDK Demo Package: A2019.07-Package-Sdk-1.0.0.zip
- Documentation: A2019.07-package-annotations-javadoc.zip

Related concepts
Automation Anywhere Enterprise Package Development Kit

